

```
1: package Models;
2:
3: import Controllers.DBSupport;
4:
5: /**
6:  * Model for the Message Table. Essentially this is what the table will contain
7:  * @Author Dylan Mrzlak
8:  */
9: public class Message {
10:
11:     private Integer id;
12:
13:     private Integer senderId;
14:
15:     private Integer wId;
16:
17:     private Integer cID;
18:
19:     private Integer recipientID;
20:
21:     private String content;
22:
23:     private Boolean pinned;
24:
25:     public Message(Integer id, Integer senderId, Integer wId, Integer cID, Integer recipientID, String content, Boolean pinned) {
26:         this.id = id;
27:         this.senderId = senderId;
28:         this.wId = wId;
29:         this.cID = cID;
30:         this.recipientID = recipientID;
31:         this.content = content;
32:         this.pinned = pinned;
33:     }
34:     //public message
35:     public Message(Integer id, Integer senderId, Integer recipientID, String content) {
36:         this.id = id;
37:         this.senderId = senderId;
38:         this.recipientID = recipientID;
39:         this.content = content;
40:     }
41:     //dm
42:     public Message(Integer id, Integer senderId, Integer wId, Integer cID, String content, Boolean pinned) {
43:         this.id = id;
44:         this.senderId = senderId;
45:         this.wId = wId;
46:         this.cID = cID;
47:         this.content = content;
48:         this.pinned = pinned;
49:     }
50:
51:     public Integer getId() {
52:         return id;
53:     }
54:
55:     public void setId(Integer id) {
56:         this.id = id;
57:     }
58:
59:     public Integer getwId() {
60:         return wId;
61:     }
62:
63:     public Integer getSenderId() {
64:         return senderId;
65:     }
66:
67:     public void setSenderId(Integer senderId) {
68:         this.senderId = senderId;
69:     }
70:
71:     public void setwId(Integer wId) {
72:         this.wId = wId;
73:     }
74:
75:     public Integer getcID() {
76:         return cID;
77:     }
78:
79:     public void setcID(Integer cID) {
80:         this.cID = cID;
81:     }
82:
83:     public Integer getRecipientID() {
84:         return recipientID;
85:     }
86:
87:     public void setRecipientID(Integer recipientID) {
88:         this.recipientID = recipientID;
89:     }
90:
91:     public String getContent() {
92:         return content;
93:     }
94:
95:     public void setContent(String content) {
96:         this.content = content;
97:     }
98:
99:     public Boolean getPinned() {
100:         return pinned;
101:     }
102:
103:     public void setPinned(Boolean pinned) {
104:         this.pinned = pinned;
105:     }
106:
107:     public static DBSupport.HTTPResponse sendMessage(String senderName, String workspaceName, String channelName, String message){
108:         DBSupport.HTTPResponse res = DBSupport.sendMessage(senderName, workspaceName, channelName, message);
109:         return res;
110:     }
111:
112:     public static DBSupport.HTTPResponse sendDirectMessage(String senderName, String receiver, String message){
113:         DBSupport.HTTPResponse res = DBSupport.sendDirectMessage(senderName, receiver, message);
114:         return res;
115:     }
116: }
```



```
1: package Models;
2:
3: import Controllers.DBSupport;
4:
5: public class Channel {
6:     private Integer id;
7:
8:     private Integer wId;
9:
10:    private String name;
11:
12:    public static DBSupport.HTTPResponse createChannel(String workspaceName,
String name) {
13:        DBSupport.HTTPResponse res = DBSupport.createChannel(workspaceName,
name);
14:        return res;
15:    }
16:
17:    public Integer getId() {
18:        return id;
19:    }
20:
21:    public void setId(Integer id) {
22:        this.id = id;
23:    }
24:
25:    public Integer getwId() {
26:        return wId;
27:    }
28:
29:    public void setwId(Integer wId) {
30:        this.wId = wId;
31:    }
32:
33:    public String getName() {
34:        return name;
35:    }
36:
37:    public void setName(String name) {
38:        this.name = name;
39:    }
40:
41:    public Channel(String name, int wID){
42:        this.name = name;
43:        this.id = -1;
44:        this.wId = wID;
45:    }
46:
47:    public Channel(String name, int wID, int id){
48:        this.name = name;
49:        this.id = id;
50:        this.wId = wID;
51:    }
52: }
53:
54:
55: }
```



```

1: package Models;
2:
3: import Controllers.DBSupport;
4:
5: public class User {
6:
7:     private String name;
8:     private String password;
9:     private Integer userId;
10:    public User(String uName, String uPassword){
11:        name = uName;
12:        password = uPassword;
13:        userId = -1;
14:    }
15:    public User(String uName, String uPassword, Integer uId){
16:        name = uName;
17:        password = uPassword;
18:        userId = uId;
19:    }
20:
21:    public Integer getUserId() {
22:        return userId;
23:    }
24:
25:    public void setUserId(Integer userId) {
26:        this.userId = userId;
27:    }
28:
29:    public String getName() {
30:        return name;
31:    }
32:
33:    public void setName(String name) {
34:        this.name = name;
35:    }
36:
37:    public String getPassword() {
38:        return password;
39:    }
40:
41:    public void setPassword(String username) {
42:        password = username;
43:    }
44:
45:    public static DBSupport.HTTPResponse createUser(String name, String pass
word){
46:        DBSupport.HTTPResponse res = DBSupport.createUser(name, password);
47:        return res;
48:    }
49:
50: }

```



```

1: package Models;
2:
3: import Controllers.DBSupport;
4: /**
5:  * Model for the Workspace within the front end. Will contain the data and methods
6:  * @Author Dylan Mrzlak
7:  */
8: public class Workspace {
9:
10:     private String name;
11:     private int id;
12:
13:     /**
14:      * Basic Constructor. Since the id is not known until it is in the DB, we
15:      * @param name
16:      * @Author Dylan Mrzlak
17:      */
18:     public Workspace(String name){
19:         this.name = name;
20:         this.id = -1;
21:     }
22:
23:     public Workspace(String name, int id){
24:         this.name = name;
25:         this.id = id;
26:     }
27:
28:     public String getName(){
29:         return name;
30:     }
31:
32:     int getId(){
33:         return id;
34:     }
35:
36:     public void setName(String name) {
37:         this.name = name;
38:     }
39:
40:     public int getId() {
41:         return id;
42:     }
43:
44:     public void setId(int id) {
45:         this.id = id;
46:     }
47:
48:     /**
49:      * Create a workspace and call for the DBSupport to request it put into
50:      * @Author Dylan Mrzlak
51:      */
52:     public static DBSupport.HTTPResponse createWorkspace(String name){
53:         DBSupport.HTTPResponse res = DBSupport.putWorkspace(name);
54:         return res;
55:     }
56:
57:     public static DBSupport.HTTPResponse joinWorkspace(String name, String username){
58:         DBSupport.HTTPResponse res = DBSupport.joinWorkspace(name, username);
59:         return res;
60:     }
61:
62:     public static DBSupport.HTTPResponse pinMessage(String mId) {
63:         DBSupport.HTTPResponse res = DBSupport.pinMessage(Integer.parseInt(m
64:         return res;
65:     }
66:
67:     public static DBSupport.HTTPResponse getUsersInWorkspace(String workspaceName){
68:         DBSupport.HTTPResponse res = DBSupport.viewUsers(workspaceName);
69:         return res;
70:     }
71: }

```





```

1: package Controllers;
2:
3: import Models.Message;
4:
5: import java.io.BufferedReader;
6: import java.io.InputStreamReader;
7: import java.io.Reader;
8: import java.net.*;
9: import java.io.IOException;
10: import java.net.MulticastSocket;
11: import java.net.URI;
12: import java.net.URISyntaxException;
13:
14: /**
15:  * This is our Data Provider for the frontend app. This will cal to the back
end controllers via HTTPRequests.
16:  * The idea goes:
17:  * User --> InputController --> Model --> DBSupport --> HTTPRequest -->
ModelController --> DB
18:  * @Author Dylan Mrzlak
19:  */
20: public class DBSupport {
21:
22:
23:     /**
24:      * Connect to the backend via HTTPRequest. The controllers on the backen
d have specific URL mappings,
25:      * so we can use those to proc the backend to do its work
26:      * @param url
27:      * @return
28:      * @throws URISyntaxException
29:      * @throws IOException
30:      * @throws InterruptedException
31:      * @Author Dylan Mrzlak
32:      */
33:     public static HTTPResponse serverRequest(String url) throws URISyntaxExc
eption, IOException, InterruptedException {
34:         //Build the URL out of the built string
35:         URL uri = new URL(url);
36:
37:         //Open a connection to uri, we will soon be able to make the actual
request
38:         //Also instantiate the rest of the request
39:         HttpURLConnection con = (HttpURLConnection) uri.openConnection();
40:         //By only using GET we can simplify calls on either side. The framew
ork we're using, spring boot doesn't care
41:         //musch about the actual request method
42:         con.setRequestMethod("GET");
43:         //We want a json to be returned in the event htat we get an object r
eturned from the controller.
44:         //They don't really send Objects, but rather a string style of encod
ing called a json.
45:         //These are really simple enough to understand when looking at the J
SONString
46:         // con.setRequestProperty("Content-Type", "application/json");
47:
48:         con.setRequestProperty("Content-Type", "application/xml");
49:         String contentType = con.getHeaderField("Content-Type");
50:
51:         //We want to know if we did good, or if Big Backend is mad at us
52:         int status = con.getResponseCode();
53:         Reader streamReader = null;
54:         //When the status is 300 or over, that means the server is not happy
to some degree about the request
55:         //Statuses usually go as follow
56:         //100 ok-ish?
57:         //200 Yay we're all good
58:         //300 Not so great
59:         //400 You messed up
60:         //500 Mr. Stark, I don't feel so good
61:         if (status > 299) {
62:             //We want to be able to read the repsonse as an Error
63:             streamReader = new InputStreamReader(con.getErrorStream());
64:         } else {
65:             //We want to be able to actually read the response
66:             streamReader = new InputStreamReader(con.getInputStream());
67:         }
68:         //Read the response, what ever it is
69:         BufferedReader in = new BufferedReader(streamReader);
70:         String inputLine;
71:         StringBuffer content = new StringBuffer();
72:         while ((inputLine = in.readLine()) != null) {
73:             content.append(inputLine);
74:         }
75:         streamReader.close();
76:         in.close();
77:         con.disconnect();
78:         //Return the response rebuilt into the HTTPResponse built here
79:         return new HTTPResponse(status, content.toString());
80:     }
81:
82:     //Prints an error for stating the a request couldn't finish for whatever
reason. Helps keep the app from being pissy
83:     public static String handleErr(){
84:         System.out.println("Unable to handle the request, please check your
connection, try again");
85:         return null;
86:     }
87:
88:     /**
89:      * Creates a request to the backend to make a Workspace
90:      * @param name
91:      * @return
92:      * @Author Dylan Mrzlak
93:      */
94:     public static HTTPResponse putWorkspace(String name){
95:         try{
96:             HTTPResponse response = serverRequest(ParamBuilder.createWorkspace
(name));
97:             return response;
98:         } catch(Exception e){
99:             return new HTTPResponse(406, handleErr());
100:         }
101:     }
102:
103:     public static HTTPResponse joinWorkspace(String workspaceName, String na
me) {
104:         try{
105:             HTTPResponse response = serverRequest(ParamBuilder.joinWorkspace
(workspaceName, name));
106:             return response;
107:         } catch(Exception e){
108:             return new HTTPResponse(406, handleErr());
109:         }
110:     }
111:
112:     public static HTTPResponse createUser(String name, String password) {
113:         try{
114:             HTTPResponse response = serverRequest(ParamBuilder.createUser(na
me, password));
115:             return response;
116:         } catch(Exception e){
117:             return new HTTPResponse(406, handleErr());
118:         }
119:     }
120:     /**
121:      * Sets a message as pinned
122:      * @param id

```

```

123:      * @return
124:      * @Author Joseph Hudson
125:      */
126:      public static HTTPResponse pinMessage(Integer id) {
127:          try{
128:              HTTPResponse response = serverRequest(ParamBuilder.pinMessage(id
));
129:              return response;
130:          } catch(Exception e){
131:              return new HTTPResponse(406, handleErr());
132:          }
133:      }
134:
135:      public static HTTPResponse sendMessage(String senderName, String workspaceName, String channelName, String message) {
136:          try{
137:              HTTPResponse response = serverRequest(ParamBuilder.sendMessage(senderName, workspaceName, channelName, message));
138:              return response;
139:          } catch(Exception e){
140:              return new HTTPResponse(406, handleErr());
141:          }
142:      }
143:
144:      public static HTTPResponse createChannel(String workspaceName, String name) {
145:          try{
146:              HTTPResponse response = serverRequest(ParamBuilder.addNewChannel(workspaceName, name));
147:              return response;
148:          } catch (Exception e){
149:              return new HTTPResponse(406, handleErr());
150:          }
151:      }
152:
153:      public static HTTPResponse sendDirectMessage(String senderName, String receiver, String message) {
154:          try{
155:              HTTPResponse response = serverRequest(ParamBuilder.sendDirectMessage(senderName, receiver, message));
156:              return response;
157:          } catch(Exception e){
158:              return new HTTPResponse(406, handleErr());
159:          }
160:      }
161:
162:      public static HTTPResponse viewUsers(String workspaceName) {
163:          try{
164:              HTTPResponse response = serverRequest(ParamBuilder.getUsersInWorkspace(workspaceName));
165:              return response;
166:          } catch (Exception e){
167:              return new HTTPResponse(406, handleErr());
168:          }
169:      }
170:
171:      /**
172:       * Model for the HTTPResponse rebuilding, that way the objects can handle the data themselves
173:       * @author Dylan Mrzlak
174:       */
175:      public static class HTTPResponse{
176:          public int code;
177:          public String response;
178:
179:          HTTPResponse(int status, String content){
180:              code = status;
181:              response = content;
182:          }
183:      }
184:
185:      /**
186:       * Static class to build our URL's to Strings.
187:       * Makes it a lot better to send it out to here, rather than build them in other methods
188:       */
189:      private static class ParamBuilder{
190:          //This is the base url for our server. When we get a dedicated server for the app, we will want this changed
191:          private static String BASE_URL = "http://localhost:8080/";
192:
193:          //Below are the Builders for the URL mappings
194:          //URLs are as follows:
195:          //      BASE_URL + CONTROLLER_MAPPING + / + REQUESTMAPPING + ?PARAM1_NAME=PARAM1
196:          //For 2+ params:
197:          //      BASE_URL + CONTROLLER_MAPPING + / + REQUESTMAPPING + ?PARAM1_NAME=PARAM1&PARAM2_NAME=PARAM2....
198:
199:          public static String sendDirectMessage(String sender, String receiver, String message){
200:              return BASE_URL+"/message/directMessage?senderName="+sender+"&receiverName="+receiver+"&message="+message;
201:          }
202:
203:          public static String sendMessage(String sender, String workspace, String channelName, String message){
204:              return BASE_URL+"/message/channelMessage?senderName="+sender+"&workspaceName="+workspace+"&channelName="+channelName+"&message="+message;
205:          }
206:
207:          public static String createWorkspace(String name){
208:              return BASE_URL+"/workspace/add?name="+name;
209:          }
210:
211:          public static String joinWorkspace(String workspaceName, String username){
212:              return BASE_URL+"/user/join?workspaceName="+workspaceName+"&name="+username;
213:          }
214:
215:          public static String createUser(String name, String password){
216:              return BASE_URL+"/user/add?username="+name+"&password="+password;
217:          }
218:
219:          public static String pinMessage(int mId){
220:              return BASE_URL+"/message/pinMessage?messageID="+ mId;
221:          }
222:
223:          public static String getUsersInWorkspace(String workspaceName) {
224:              return BASE_URL+"/workspace/getUsers/?name="+workspaceName;
225:          }
226:
227:          public static String addNewChannel(String workspaceName, String name) {
228:              return BASE_URL+"/channel/add/?name="+workspaceName+"&name="+name;
229:          }
230:      }

```

```

1: import Controllers.DBSupport;
2: import Models.Message;
3: import Models.Channel;
4: import Models.User;
5: import Models.Workspace;
6:
7: import java.util.Random;
8: import java.util.Scanner;
9: import com.google.gson.Gson;
10:
11:
12: /**
13:  * This will be the main controller for the application.
14:  *
15:  * It will take the initial input for User Input
16:  * and then pass it along to other classes to handle the actual functional
ty
17:  * @Author Dylan Mrzlak
18:  * Original Framework and Use handle for CREATE_WORKSPACE and JOIN_WORK
SPACE
19:  */
20: public class InputController {
21:     private static final String CREATE_WORKSPACE = "create workspace";
22:     private static final String JOIN_WORKSPACE = "join";
23:     private static final String CREATE_CHANNEL = "create channel";
24:     private static final String VIEW_USERS = "view users";
25:     private static final String SEND = "send";
26:     private static final String SEND_DM = "send to";
27:     private static final String ADD_USER = "create user";
28:     private static final String LOGIN = "login";
29:     private static final String PIN_MESSAGE = "pin message";
30:
31:     public static void main(String[] args){
32:         //If this line get mad, check your dependencies, may have dropped
33:         Gson gson = new Gson();
34:         User thisUser = null;
35:         Workspace cur = null;
36:         Scanner input = new Scanner(System.in);
37:         String userInput = "";
38:         Channel curChannel = null;
39:
40:         /*
41:          * We want to handle all forms of input via commands. That is everythin
g is in the switch case.
42:          * Eventually, logan or I will get HELP commands in here. Probably jus
t listing the commands and arguments
43:          * Possisble allowing "HELP - COMMAND" which will explain what the comm
and does and the arguments/preconditions
44:          *
45:          *
46:          *
47:          *
48:          *
49:          System.out.println("Please enter username:\n");
50:          User current = new User("", "");
51:          System.out.println("Please enter your password!\n");
52:          String password = input.nextLine();
53:          */
54:
55:         do {
56:             userInput = input.nextLine();
57:             //By forcing commands to be in a format of COMMAND - ARGUMENT
58:             //We can easily manage the input and decide what is needed
59:             int substringBegin = userInput.indexOf('-');
60:             if(substringBegin == -1) substringBegin = 0;
61:             String command = userInput.substring(0, substringBegin).trim();
62:             String[] userArgs = userInput.substring(substringBegin + 1).trim()
.split(" ");
63:             switch (command) {
64:
65:                 case ADD_USER:
66:                     if(userArgs.length != 2) {
67:                         System.out.println("Invalid Number or Arguments");
68:                         break;
69:                     }
70:                     DBSupport.HTTPResponse uResponse = User.createUser(userArg
s[0], userArgs[1]);
71:
72:                     if (uResponse.code > 300) {
73:                         System.out.println(uResponse.response);
74:                     } else {
75:                         System.out.println("Saved User");
76:                         User u = gson.fromJson(uResponse.response, User.class)
77:
78:                         thisUser = u;
79:                     }
80:                     break;
81:                 case CREATE_WORKSPACE:
82:                     if(userArgs.length != 1) {
83:                         System.out.println("Invalid Number or Arguments");
84:                         break;
85:                     }
86:                     DBSupport.HTTPResponse wResponse = Workspace.createWorkspa
ce(userArgs[0]);
87:
88:                     if (wResponse.code > 300) {
89:                         System.out.println(wResponse.response);
90:                     } else {
91:                         System.out.println("Saved Workspace");
92:                         Workspace w = gson.fromJson(wResponse.response, Workspa
ce.class);
93:
94:                         cur = w;
95:                     }
96:                     break;
97:                 case JOIN_WORKSPACE:
98:                     if(thisUser == null) {
99:                         System.out.println("You need to create a user or sign
in to continue");
100:                     }
101:                     break;
102:                     if(userArgs.length != 1) {
103:                         System.out.println("Invalid Number or Arguments");
104:                         break;
105:                     }
106:                     DBSupport.HTTPResponse joinWorkspace = Workspace.joinWorks
pace(userArgs[0], thisUser.getName());
107:
108:                     if (joinWorkspace.code > 300) {
109:                         System.out.println(joinWorkspace.response);
110:                     } else {
111:                         System.out.println("Joining Workspace");
112:                         Workspace w = gson.fromJson(joinWorkspace.response, Wo
rkspace.class);
113:
114:                         cur = w;
115:                     }
116:                     break;
117:                 case CREATE_CHANNEL:
118:                     if (cur == null) {
119:                         System.out.println("User not in workspace");
120:                         break;
121:                     }
122:                     if (userArgs.length != 2) {
123:                         System.out.println("Wrong Number of arguments. Try: cr
eate channel - <workspace> <name> ");
124:                     }
125:                     DBSupport.HTTPResponse cResponse = Channel.createChannel(u
serArgs[0], userArgs[1]);
126:
127:                     if (cResponse.code > 300) {
128:                         System.out.println(cResponse.response);
129:                     } else {
130:                         System.out.println("Saved Channel");
131:                         Channel c = gson.fromJson(cResponse.response, Channel.

```

```

class;
124:         curChannel = c;
125:     }
126:     break;
127:     case VIEW_USERS:
128:         if (cur == null) {
129:             System.out.println("User not in workspace");
130:             break;
131:         }
132:         DBSupport.HTTPResponse viewUsers = Workspace.getUsersInWor
kspace(cur.getName());
133:         if(viewUsers.code > 300) {
134:             System.out.println("There are no users in this workspa
ce");
135:         }
136:         String[] userList = gson.fromJson(viewUsers.response, Stri
ng[].class);
137:         System.out.println("\nUsers in workspace: " + cur.getName(
));
138:         for(int i = 0; i < userList.length; i++){
139:             System.out.println("\t"+userList[i]);
140:         }
141:         System.out.println("\n");
142:         break;
143:     case PIN_MESSAGE:
144:         if(thisUser == null) {
145:             System.out.println("You need to create a user or sign
in to continue");
146:             break;
147:         }
148:         if(userArgs.length != 1) {
149:             System.out.println("Invalid Number or Arguments");
150:             break;
151:         }
152:         DBSupport.HTTPResponse pinMessage = Workspace.pinMessage(u
serArgs[0]);
153:         if (pinMessage.code > 300) {
154:             System.out.println(pinMessage.response);
155:         }
156:         else {
157:             System.out.println("Pinned message");
158:             Message m = gson.fromJson(pinMessage.response, Message
.class);
159:         }
160:         break;
161:     case SEND_DM:
162:         if(userArgs.length < 2){
163:             System.out.println("Invalid number of arguments");
164:         }
165:         String directMessage = "";
166:         for(int i = 1; i < userArgs.length; i++){
167:             directMessage += userArgs[i];
168:         }
169:         DBSupport.HTTPResponse dm = Message.sendDirectMessage(this
User.getName(), userArgs[0], directMessage);
170:         if (dm.code > 300) {
171:             System.out.println(dm.response);
172:         } else {
173:             System.out.println("Joining Workspace");
174:             Message m = gson.fromJson(dm.response, Message.class);
175:             System.out.println("Message Sent: \n\t" + m.getContent
());
176:         }
177:         break;
178:     case SEND:
179:         if(thisUser == null) {
180:             System.out.println("You need to create a user or sign
in to continue");
181:             break;
182:         }
183:         if (cur == null) {
184:             System.out.println("User not in workspace");
185:             break;
186:         }
187:         if (curChannel == null) {
188:             System.out.println("User not in Channel;");
189:             break;
190:         }
191:         if(userArgs.length < 1){
192:             System.out.println("Invalid number of arguments");
193:         }
194:         String message = "";
195:         for(int i = 0; i < userArgs.length; i++){
196:             message += userArgs[i];
197:         }
198:         DBSupport.HTTPResponse sendMessage = Message.sendMessage(t
hisUser.getName(), curChannel.getName(), message);
199:         if (sendMessage.code > 300) {
200:             System.out.println(sendMessage.response);
201:         } else {
202:             System.out.println("Joining Workspace");
203:             Message m = gson.fromJson(sendMessage.response, Messag
e.class);
204:             System.out.println("Message Sent: \n\t" + m.getContent
());
205:         }
206:         break;
207:         default:
208:             System.out.println("Invalid Input please try again :(");
209:             break;
210:         }
211:     } while (input.hasNextLine());
212: }
213: }
214:
215:
216: }

```

```
1: package com.slack.server;
2:
3: import org.junit.jupiter.api.Test;
4: import org.springframework.boot.test.context.SpringBootTest;
5:
6: @SpringBootTest
7: class ServerApplicationTests {
8:
9:     @Test
10:     void contextLoads() {
11:     }
12:
13: }
```



```

1: package com.slack.server.messages;
2:
3: import com.slack.server.channel.Channel;
4: import com.slack.server.channel.ChannelRepository;
5: import com.slack.server.user.User;
6: import com.slack.server.user.UserRepository;
7: import com.slack.server.workspace.Workspace;
8: import com.slack.server.workspace.WorkspaceRepository;
9: import com.slack.server.workspaceXRef.WorkspaceXRefRepository;
10: import org.springframework.beans.factory.annotation.Autowired;
11: import org.springframework.http.HttpStatus;
12: import org.springframework.http.ResponseEntity;
13: import org.springframework.stereotype.Controller;
14: import org.springframework.web.bind.annotation.RequestMapping;
15: import org.springframework.web.bind.annotation.RequestParam;
16: import org.springframework.web.bind.annotation.ResponseBody;
17:
18: @Controller // This means that this class is a Controller
19: @RequestMapping(path="/message") // This means URL's start with /demo (after
Application path)
20: public class MessageController {
21:
22:     @Autowired
23:     private MessageRepository messageRepository;
24:     @Autowired
25:     private WorkspaceRepository workspaceRepository;
26:     @Autowired
27:     private UserRepository userRepository;
28:     @Autowired
29:     private ChannelRepository channelRepository;
30:     @Autowired
31:     private WorkspaceXRefRepository workspaceXRefRepository;
32:
33:
34:
35:     @RequestMapping(path="/directMessage")
36:     @ResponseBody
37:     ResponseEntity directMessage(@RequestParam String senderName, @RequestPa
ram String recieverName, @RequestParam String message){
38:         User sender = userRepository.findByName(senderName);
39:         if(sender == null)
40:             return new ResponseEntity("User Sender Not Found!!!", HttpStatus
.NOT_FOUND);
41:         User recipient = userRepository.findByName(recieverName);
42:         if(recipient == null)
43:             return new ResponseEntity("User recipient Not Found!!!", HttpSta
tus.NOT_FOUND);
44:         Message m = new Message();
45:         m.setSenderId(sender.getId());
46:         m.setRecipientID(recipient.getId());
47:         m.setContent(message);
48:         m.setwId(null);
49:         m.setcId(null);
50:         m.setPinned(false);
51:         messageRepository.save(m);
52:         return new ResponseEntity(m, HttpStatus.OK);
53:     }
54:
55:     @RequestMapping(path="/channelMessage")
56:     @ResponseBody
57:     ResponseEntity channelMessage(@RequestParam String senderName, @RequestP
aram String workSpaceName,
58:                                   @RequestParam String channelName, @Request
Param String message){
59:         User sender = userRepository.findByName(senderName);
60:         if(sender == null)
61:             return new ResponseEntity("User Sender Not Found!!!", HttpStatus
.NOT_FOUND);
62:         Workspace workspace = workspaceRepository.findbyName(workSpaceName);
63:
64:         if(workspace == null)
65:             return new ResponseEntity("Workspace Not Found!!", HttpStatus.NO
T_FOUND);
66:         Channel channel = channelRepository.find(workspace.getId(), channelNa
me);
67:         if(channel == null)
68:             return new ResponseEntity("Channel Not Found :", HttpStatus.NOT
_FOUND);
69:         Message m = new Message();
70:         m.setSenderId(sender.getId());
71:         m.setRecipientID(null);
72:         m.setContent(message);
73:         m.setwId(workspace.getId());
74:         m.setcId(channel.getId());
75:         m.setPinned(false);
76:         messageRepository.save(m);
77:         return new ResponseEntity(m, HttpStatus.OK);
78:     }
79:
80:     /**
81:      * sets a messages pinned status to true
82:      * @param messageID
83:      * @return error if ID doesn't match any existing messages.
84:      * @author Joseph Hudson
85:      */
86:     @RequestMapping(path="/pinMessage")
87:     @ResponseBody
88:     ResponseEntity pinMessage(@RequestParam Integer messageID){
89:         if(!messageRepository.existsById(messageID)) return new ResponseEnti
ty("No message with that ID is found", HttpStatus.NOT_ACCEPTABLE);
90:         Message m = messageRepository.findById(messageID);
91:         m.setPinned(true);
92:         messageRepository.save(m);
93:         return new ResponseEntity(m, HttpStatus.OK);
94:     }
95:
96:
97:
98:

```





```

1: package com.slack.server.messages;
2:
3:
4: import org.springframework.lang.Nullable;
5:
6: import javax.persistence.*;
7: /**
8:  * Model for the Message Table. Essentially this is what the table will contain
9:  * @Author Dylan Mrzlak
10:  */
11: @Entity
12: public class Message {
13:
14:     @Id
15:     @GeneratedValue(strategy= GenerationType.AUTO)
16:     private Integer id;
17:
18:     private Integer senderId;
19:
20:     @Nullable
21:     private Integer wId;
22:
23:     private Integer cID;
24:
25:     private Integer recipientID;
26:
27:     private String content;
28:
29:     private Boolean pinned;
30:
31:     public Integer getId() {
32:         return id;
33:     }
34:
35:     public void setId(Integer id) {
36:         this.id = id;
37:     }
38:
39:     public Integer getwId() {
40:         return wId;
41:     }
42:
43:     public Integer getSenderId() {
44:         return senderId;
45:     }
46:
47:     public void setSenderId(Integer senderId) {
48:         this.senderId = senderId;
49:     }
50:
51:     public void setwId(Integer wId) {
52:         this.wId = wId;
53:     }
54:
55:     public Integer getcID() {
56:         return cID;
57:     }
58:
59:     public void setcID(Integer cID) {
60:         this.cID = cID;
61:     }
62:
63:     public Integer getRecipientID() {
64:         return recipientID;
65:     }
66:
67:     public void setRecipientID(Integer recipientID) {
68:         this.recipientID = recipientID;
69:     }
70:
71:     public String getContent() {
72:         return content;
73:     }
74:
75:     public void setContent(String content) {
76:         this.content = content;
77:     }
78:
79:     public Boolean getPinned() {
80:         return pinned;
81:     }
82:
83:     public void setPinned(Boolean pinned) {
84:         this.pinned = pinned;
85:     }
86: }

```



```

1: package com.slack.server.messages;
2:
3: import org.springframework.data.jpa.repository.Query;
4: import org.springframework.data.repository.CrudRepository;
5: import org.springframework.data.repository.query.Param;
6:
7: public interface MessageRepository extends CrudRepository<Message, Integer>
{
8:
9:     @Query("SELECT CASE WHEN COUNT(m) > 0 THEN true ELSE false END FROM Mess
age m WHERE m.id = :id")
10:     boolean existsById(@Param("id") Integer id);
11:
12:     @Query("SELECT m FROM Message m WHERE m.id = :id")
13:     Message findById(@Param("id") Integer id);
14:
15:     @Query("Select m From Message m where m.wId = :wId AND m.cId = :cId")
16:     Iterable<Message> getChannelMessages(@Param("wId") int wId, @Param("cId"
) int cId);
17:
18:     @Query("Select m From Message m where m.recipientID = :rId")
19:     Iterable<Message> getUsersMessages(@Param("rId") int rId);
20:
21: }

```



```

1: package com.slack.server.workspace;
2:
3: import com.slack.server.user.User;
4: import jdk.nashorn.internal.objects.annotations.Property;
5: import org.springframework.data.jpa.repository.JpaRepository;
6: import org.springframework.data.jpa.repository.Query;
7: import com.slack.server.workspace.Workspace;
8: import org.springframework.data.jpa.repository.query.Procedure;
9: import org.springframework.data.repository.CrudRepository;
10: import org.springframework.data.repository.query.Param;
11:
12: import javax.persistence.NamedStoredProcedureQueries;
13: import javax.persistence.NamedStoredProcedureQuery;
14: import javax.persistence.ParameterMode;
15: import javax.persistence.StoredProcedureParameter;
16:
17: public interface WorkspaceRepository extends CrudRepository<Workspace, Integer>{
18:
19:     @Query("SELECT CASE WHEN COUNT(w) > 0 THEN true ELSE false END FROM Workspace w WHERE w.name = :name")
20:     boolean existsByName(@Param("name") String name);
21:
22:     @Query("SELECT w FROM Workspace w WHERE w.name = :name")
23:     Workspace findByName(@Param("name") String name);
24:
25: }
26:
27:

```



```

1: package com.slack.server.workspace;
2:
3: import com.slack.server.user.User;
4: import com.slack.server.user.UserRepository;
5: import org.springframework.beans.factory.annotation.Autowired;
6: import org.springframework.http.HttpStatus;
7: import org.springframework.http.ResponseEntity;
8: import org.springframework.stereotype.Controller;
9: import org.springframework.web.bind.annotation.GetMapping;
10: import org.springframework.web.bind.annotation.RequestMapping;
11: import org.springframework.web.bind.annotation.RequestParam;
12: import org.springframework.web.bind.annotation.ResponseBody;
13:
14: import java.util.HashMap;
15:
16: @Controller // This means that this class is a Controller
17: @RequestMapping(path="/workspace") // This means URL's start with /demo (after Application path)
18: public class WorkspaceController {
19:     // This means to get the bean called userRepository
20:     // Which is auto-generated by Spring, we will use it to handle the data
21:     @Autowired
22:     private WorkspaceRepository workspaceRepository;
23:
24:     @Autowired
25:     private UserRepository userRepository;
26:
27:     /**
28:      * Create and put a Workspace into the table
29:      * @param name
30:      * @return
31:      * @author Dylan Mrzlak
32:      */
33:     @GetMapping(path="/add") // Map ONLY POST Requests
34:     public @ResponseBody ResponseEntity addNewWorkspace (@RequestParam String name) {
35:         // @ResponseBody means the returned String is the response, not a view
36:         // @RequestParam means it is a parameter from the GET or POST request
37:         if(workspaceRepository.existsByName(name)) return new ResponseEntity
("Workspace name already taken", HttpStatus.NOT_ACCEPTABLE);
38:         Workspace n = new Workspace();
39:         n.setName(name);
40:         workspaceRepository.save(n);
41:         return new ResponseEntity(n, HttpStatus.OK);
42:     }
43:
44:     /**
45:      * Get all workspaces in DB
46:      * @return
47:      * @Author Dylan Mrzlak
48:      */
49:     @GetMapping(path="")
50:     public @ResponseBody ResponseEntity getAllWorkspaces() {
51:         // This returns a JSON or XML with the workspaces
52:         Iterable<Workspace> list = workspaceRepository.findAll();
53:         return new ResponseEntity(list, HttpStatus.OK);
54:     }
55:
56:     /**
57:      * Get a workspace by name from the DB
58:      * @param name
59:      * @return
60:      * @Author Dylan Mrzlak
61:      */
62:     @GetMapping(path="/get")
63:     public @ResponseBody ResponseEntity getWorkspaceByName (@RequestParam String name) {
64:         if(workspaceRepository.existsByName(name)) return new ResponseEntity
(workspaceRepository.findByName(name), HttpStatus.OK);
65:         return new ResponseEntity("Workspace does not exist", HttpStatus.NOT_FOUND);
66:     }
67:
68:     @GetMapping(path="/getUsers")
69:     public @ResponseBody ResponseEntity getUsersInWorkspace (@RequestParam String name) {
70:         Iterable<String> list = userRepository.findUsers(name);
71:         return new ResponseEntity(list, HttpStatus.OK);
72:     }
73:
74:
75: }

```





```

1: package com.slack.server.workspace;
2:
3: import com.slack.server.channel.Channel;
4: import com.slack.server.user.User;
5: import org.hibernate.validator.constraints.UniqueElements;
6:
7: import javax.persistence.*;
8: import java.util.Set;
9:
10: /**
11:  * Model for the Workspace Table. Essentially this is what the table will co
ntain
12:  * @Author Dylan Mrzlak
13:  */
14: @Entity // This tells Hibernate to make a table out of this class
15: public class Workspace {
16:     @Id
17:     @GeneratedValue(strategy=GenerationType.AUTO)
18:     private Integer id;
19:
20:     private String name;
21:
22:     public Integer getId() {
23:         return id;
24:     }
25:
26:     public void setId(Integer id) {
27:         this.id = id;
28:     }
29:
30:     public String getName() {
31:         return name;
32:     }
33:
34:     public void setName(String name) {
35:         this.name = name;
36:     }
37:
38: }

```



```

1: package com.slack.server.user;
2:
3: import org.springframework.data.jpa.repository.JpaRepository;
4: import org.springframework.data.jpa.repository.Query;
5: import com.slack.server.user.User;
6: import org.springframework.data.jpa.repository.query.Procedure;
7: import org.springframework.data.repository.CrudRepository;
8: import org.springframework.data.repository.query.Param;
9:
10: import java.util.HashMap;
11:
12: public interface UserRepository extends CrudRepository<User, Integer>{
13:     @Query("SELECT CASE WHEN COUNT(u) > 0 THEN true ELSE false END FROM User
u WHERE u.name = :name")
14:     boolean existsByName(@Param("name") String name);
15:
16:     @Query("SELECT u FROM User u WHERE u.name = :name")
17:     User findByName(@Param("name") String name);
18:
19:     @Query("Select u.name "+
20:           "From User u Left Join WorkspaceXRef x on u.id = x.uId "+
21:           "where x.wId = (select id from Workspace w where w.name = :wName
)")
22:     Iterable<String> findUsers(@Param("wName") String name);
23:
24: }

```



```

1: package com.slack.server.user;
2: import com.slack.server.channel.Channel;
3: import org.hibernate.validator.constraints.UniqueElements;
4:
5: import javax.persistence.*;
6:
7: /**
8:  * Model for the User Table. Essentially this is what the table will contain
9:  * @Author Dylan Mrzlak
10: */
11: @Entity // This tells Hibernate to make a table out of this class
12: public class User {
13:
14:     @Id
15:     @GeneratedValue(strategy=GenerationType.SEQUENCE)
16:     private Integer id;
17:
18:     private String name;
19:
20:     private String password;
21:
22:     public Integer getId() {
23:         return id;
24:     }
25:
26:     public void setId(Integer id) {
27:         this.id = id;
28:     }
29:
30:     public String getName() {
31:         return name;
32:     }
33:
34:     public void setName(String name) {
35:         this.name = name;
36:     }
37:
38:     public String getPassword(){ return password;};
39:
40:     public void setPassword(String password){this.password = password; }
41:
42: }

```



```

1: package com.slack.server.user;
2:
3: import com.slack.server.workspace.Workspace;
4: import com.slack.server.workspace.WorkspaceRepository;
5: import com.slack.server.workspaceXRef.WorkspaceXRef;
6: import com.slack.server.workspaceXRef.WorkspaceXRefRepository;
7: //import javax.util.Pair;
8: import org.springframework.beans.factory.annotation.Autowired;
9: import org.springframework.http.HttpStatus;
10: import org.springframework.http.ResponseEntity;
11: import org.springframework.stereotype.Controller;
12: import org.springframework.web.bind.annotation.GetMapping;
13: import org.springframework.web.bind.annotation.RequestMapping;
14: import org.springframework.web.bind.annotation.RequestParam;
15: import org.springframework.web.bind.annotation.ResponseBody;
16:
17: @Controller // This means that this class is a Controller
18: @RequestMapping(path="/user") // This means URL's start with /demo (after Ap
plication path)
19: public class UserController {
20:
21:     @Autowired
22:     private UserRepository uRepo;
23:
24:     @Autowired
25:     private WorkspaceRepository wRepo;
26:
27:     @Autowired
28:     private WorkspaceXRefRepository wXRefRepo;
29:
30:     /**
31:      * Create a user for the DB and put them into the table
32:      * @param username
33:      * @param password
34:      * @return
35:      * @author Dylan Mrzlak
36:      */
37:     @GetMapping(path="/add") // Map ONLY POST Requests
38:     public @ResponseBody ResponseEntity createUser(@RequestParam String user
name, @RequestParam String password){
39:         if(uRepo.existsByName(username)) return new ResponseEntity("Username
is taken", HttpStatus.NOT_ACCEPTABLE);
40:         User u = new User();
41:         u.setName(username);
42:         u.setPassword(password);
43:         uRepo.save(u);
44:         return new ResponseEntity(u, HttpStatus.OK);
45:     }
46:
47:     @GetMapping(path="/login")
48:     public @ResponseBody ResponseEntity login(@RequestParam String username,
@RequestParam String password){
49:         if(!uRepo.existsByName(username)) return new ResponseEntity("No User
found", HttpStatus.NOT_FOUND);
50:         User u = uRepo.findByName(username);
51:         if(password.equals(u.getPassword())) return new ResponseEntity(u, Ht
tpStatus.OK);
52:         return new ResponseEntity("Incorrect Password", HttpStatus.NOT_ACCEP
TABLE);
53:     }
54:
55:     /**
56:      * Gets all the users in the DB
57:      * @return
58:      * @author Dylan
59:      */
60:     @GetMapping(path="")
61:
62:
63:     public @ResponseBody ResponseEntity getAllUsers() {
64:         // This returns a JSON or XML with the workspaces
65:         Iterable<User> list = uRepo.findAll();
66:         return new ResponseEntity(list, HttpStatus.OK);
67:     }
68:
69:     @GetMapping(path="/get")
70:     public @ResponseBody ResponseEntity getUser(@RequestParam String name){
71:         if(uRepo.existsByName(name)){
72:             User u = uRepo.findByName(name);
73:             return new ResponseEntity(u, HttpStatus.OK);
74:         }
75:         return new ResponseEntity("User not found", HttpStatus.NOT_FOUND);
76:     }
77:
78:     /**
79:      * When accessed, will add a user to a workspace. In the workspaceXRef t
able, if a user and a workspace are in the
80:      * same row, then that user is in the workspace
81:      * @param workspaceName
82:      * @param name
83:      * @return
84:      * @author Dylan Mrzlak
85:      */
86:     @GetMapping(path="/join")
87:     public @ResponseBody ResponseEntity joinWorkspace(@RequestParam String w
orkspaceName, @RequestParam String name){
88:         //Get Workspace, we will use its ID later
89:         Workspace w = wRepo.findbyName(workspaceName);
90:         if(w == null) return new ResponseEntity("Workspace not found", HttpS
tatus.NOT_FOUND);
91:         //Get the user, we will use their ID later
92:         User u = uRepo.findByName(name);
93:         if(u == null) return new ResponseEntity("User not found", HttpStatu
s.NOT_FOUND);
94:         //Chack that the user isn't already in the workspace
95:         if(wXRefRepo.exists(w.getId(), u.getId())) return new ResponseEntity
("User already in Workspace", HttpStatus.NOT_ACCEPTABLE);
96:         //Create the XREF and put it in DB
97:         WorkspaceXRef x = new WorkspaceXRef();
98:         x.setwId(w.getId());
99:         x.setuId(u.getId());
100:         wXRefRepo.save(x);
101:
102:         //Return OK status (200) and workspace
103:         return new ResponseEntity(w, HttpStatus.OK);
104:     }
105:
106: }
107:
108: }

```





```
1: package com.slack.server.workspaceXRef;
2:
3: import org.springframework.data.jpa.repository.Query;
4: import org.springframework.data.repository.CrudRepository;
5: import org.springframework.data.repository.query.Param;
6:
7: public interface WorkspaceXRefRepository extends CrudRepository<WorkspaceXRef, Integer> {
8:     @Query("SELECT CASE WHEN COUNT(x) > 0 THEN true ELSE false END FROM WorkspaceXRef x WHERE x.wId = :wId AND x.uId = :uId")
9:     boolean exists(@Param("wId") int wId, @Param("uId") int uId);
10: }
```



```

1: package com.slack.server.workspaceXRef;
2:
3: import javax.persistence.*;
4:
5:
6: /**
7:  * Model for the WorkspaceXRef Table. Essentially this is what the table will
l contain
8:  * If a row exists in this table the User with uID belongs to the workspace
with wId
9:  * @Author Dylan Mrzlak
10:  */
11: @Entity
12: public class WorkspaceXRef {
13:     //We will use this table to represent a user being a part of a workspace
.
14:     //If a user's ID exists in this table with a workspace's ID, then that u
ser is in that worksapce
15:
16:     @Id
17:     @GeneratedValue(strategy= GenerationType.AUTO)
18:     private Integer id;
19:
20:     private Integer wId;
21:     private Integer uId;
22:
23:     public Integer getId() {
24:         return id;
25:     }
26:
27:     public void setId(Integer id) {
28:         this.id = id;
29:     }
30:
31:     public int getuId() {
32:         return uId;
33:     }
34:
35:     public int getwId() {
36:         return wId;
37:     }
38:
39:     public void setuId(int uId) {
40:         this.uId = uId;
41:     }
42:
43:     public void setwId(int wId) {
44:         this.wId = wId;
45:     }
46: }

```



```
1: package com.slack.server;
2:
3: import org.springframework.boot.SpringApplication;
4: import org.springframework.boot.autoconfigure.SpringBootApplication;
5:
6: @SpringBootApplication
7: public class ServerApplication {
8:
9:     public static void main(String[] args) {
10:         SpringApplication.run(ServerApplication.class, args);
11:     }
12:
13: }
```



```

1: package com.slack.server.channel;
2:
3: import com.slack.server.workspace.Workspace;
4: import com.slack.server.workspace.WorkspaceRepository;
5: //import javax.xml.bind.annotation.XmlRootElement;
6: import org.springframework.beans.factory.annotation.Autowired;
7: import org.springframework.http.HttpStatus;
8: import org.springframework.http.ResponseEntity;
9: import org.springframework.stereotype.Controller;
10: import org.springframework.web.bind.annotation.GetMapping;
11: import org.springframework.web.bind.annotation.PostMapping;
12: import org.springframework.web.bind.annotation.RequestMapping;
13: import org.springframework.web.bind.annotation.RequestParam;
14: import org.springframework.web.bind.annotation.ResponseBody;
15:
16: @Controller // This means that this class is a Controller
17: @RequestMapping(path="/channel") // This means URL's start with /demo (after
Application path)
18: public class ChannelController {
19:
20:     @Autowired
21:     private WorkspaceRepository workspaceRepository;
22:
23:     @Autowired
24:     private ChannelRepository channelRepository;
25:
26:     /**
27:      * Create a channel for the DB and put them into the table
28:      * @param workspaceName
29:      * @param name
30:      * @return
31:      * @author Dylan Mrzlak
32:      */
33:     @GetMapping(path="/add") // Map ONLY POST Requests
34:     public @ResponseBody ResponseEntity addNewChannel(@RequestParam String w
orkspaceName, @RequestParam String name){
35:         Workspace w = workspaceRepository.findbyName(workspaceName);
36:         if(w == null) return new ResponseEntity("Workspace not found", HttpSt
atus.NOT_FOUND);
37:         if(channelRepository.exists(w.getId(), name)) return new ResponseEnt
ity("Channel Already Exists", HttpStatus.NOT_ACCEPTABLE);
38:         Channel c = new Channel();
39:         c.setwId(w.getId());
40:         c.setName(name);
41:         channelRepository.save(c);
42:         return new ResponseEntity(c, HttpStatus.OK);
43:     }
44:
45:     /**
46:      * Gets all the Channels in the DB
47:      * @return
48:      * @author Dylan
49:      */
50:     @GetMapping(path="")
51:     public @ResponseBody ResponseEntity getAllChannels() {
52:         // This returns a JSON or XML with the workspaces
53:         Iterable<Channel> list = channelRepository.findAll();
54:         return new ResponseEntity(list, HttpStatus.OK);
55:     }
56:
57:
58:     /**
59:      * Gets a certain Channel in the DB
60:      * @param workspaceName
61:      * @param name
62:      * @return
63:      * @author Dylan
64:      */
65:     @GetMapping(path="/get")
66:     public @ResponseBody ResponseEntity getChannel(@RequestParam String work
spaceName, @RequestParam String name){
67:         //Check that the workspace itself exists
68:         Workspace w = workspaceRepository.findbyName(workspaceName);
69:         if(w == null) return new ResponseEntity("Workspace not found", HttpSt
atus.NOT_FOUND);
70:         //Return the channel and HttpStatus.200 if it exists, or a 404 and a
detail
71:         if(channelRepository.exists(w.getId(), name)){
72:             Channel c = channelRepository.find(w.getId(), name);
73:             return new ResponseEntity(c, HttpStatus.OK);
74:         }
75:         return new ResponseEntity("Channel Does Not Exist", HttpStatus.NOT_F
OUND);
76:     }
77: }
78: }

```





```

1: package com.slack.server.channel;
2:
3: import org.hibernate.validator.constraints.UniqueElements;
4:
5: import javax.persistence.*;
6:
7: /**
8:  * Model for the Channel Table. Essentially this is what the table will contain
ain
9:  * @Author Dylan Mrzlak
10:  */
11: @Entity // This tells Hibernate to make a table out of this class
12: public class Channel {
13:
14:     @Id
15:     @GeneratedValue(strategy=GenerationType.AUTO)
16:     private Integer id;
17:
18:     private Integer wId;
19:
20:     private String name;
21:
22:     public Integer getId() {
23:         return id;
24:     }
25:
26:     public void setId(Integer id) {
27:         this.id = id;
28:     }
29:
30:     public Integer getwId() {
31:         return wId;
32:     }
33:
34:     public void setwId(Integer wId) {
35:         this.wId = wId;
36:     }
37:
38:     public String getName() {
39:         return name;
40:     }
41:
42:     public void setName(String name) {
43:         this.name = name;
44:     }
45:
46: }

```



```

1: package com.slack.server.channel;
2:
3: import org.springframework.data.jpa.repository.JpaRepository;
4: import org.springframework.data.jpa.repository.Query;
5: import com.slack.server.workspace.Workspace;
6: import org.springframework.data.repository.CrudRepository;
7: import org.springframework.data.repository.query.Param;
8:
9: public interface ChannelRepository extends CrudRepository<Channel, Integer>{
10:
11:     @Query("SELECT CASE WHEN COUNT(c) > 0 THEN true ELSE false END FROM Chan
nel c WHERE c.wId = :wId AND c.name = :name")
12:     boolean exists(@Param("wId") int wId, @Param("name") String name);
13:
14:     @Query("SELECT c FROM Channel c WHERE c.wId = :wId AND c.name = :name")
15:     Channel find(@Param("wId") int wId, @Param("name") String name);
16: }
17:
18:

```