```
    1: package Models;
    2:
    3: import Controllers.DBSupport;
    4:
    5: /**
    6:  * Model for the Message Table. Essentially this is what the table will cont
ain
    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, I
nteger 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:
   35:     //dm
   36:     public Message(Integer id, Integer senderId, Integer recipientID, String
 content) {
   37:         this.id = id;
   38:         this.senderId = senderId;
   39:         this.recipientID = recipientID;
   40:         this.content = content;
   41:     }
   42:
   43:     //public message
   44:     public Message(Integer id, Integer senderId, Integer wId, Integer cID, S
tring content, Boolean pinned) {
   45:         this.id = id;
   46:         this.senderId = senderId;
   47:         this.wId = wId;
   48:         this.cID = cID;
   49:         this.content = content;
   50:         this.pinned = pinned;
   51:     }
   52:
   53:     public static DBSupport.HTTPResponse getAllMessages(String name) {
   54:         DBSupport.HTTPResponse res = DBSupport.getAllMessages(name);
   55:         return res;
   56:     }
   57:
   58:     public Integer getId() {
   59:         return id;
   60:     }
   61:
   62:     public void setId(Integer id) {
   63:         this.id = id;
   64:     }
   65:
   66:     public Integer getwId() {
   67:         return wId;
   68:     }
   69:
   70:     public Integer getSenderId() {
   71:         return senderId;
   72:     }
   73:
   74:     public void setSenderId(Integer senderId) {
   75:         this.senderId = senderId;
   76:     }
   77:
   78:     public void setwId(Integer wId) {
   79:         this.wId = wId;
   80:     }
   81:
   82:     public Integer getcID() {
   83:         return cID;
   84:     }
   85:
   86:     public void setcID(Integer cID) {
   87:         this.cID = cID;
   88:     }
   89:
   90:     public Integer getRecipientID() {
   91:         return recipientID;
   92:     }
   93:
   94:     public void setRecipientID(Integer recipientID) {
   95:         this.recipientID = recipientID;
   96:     }
   97:
   98:     public String getContent() {
   99:         return content;
  100:     }
  101:
  102:     public void setContent(String content) {
  103:         this.content = content;
  104:     }
  105:
  106:     public Boolean getPinned() {
  107:         return pinned;
  108:     }
  109:
  110:     public void setPinned(Boolean pinned) {
  111:         this.pinned = pinned;
  112:     }
  113:
  114:     /**
  115:      * Calls DBSupport and returns the response
  116:      * @param senderName
  117:      * @param workspaceName
  118:      * @param channelName
  119:      * @param message
  120:      * @return
  121:      */
  122:     public static DBSupport.HTTPResponse sendMessage(String senderName, Stri
ng workspaceName, String channelName, String message){
  123:             DBSupport.HTTPResponse res = DBSupport.sendMessage(senderName, w
orkspaceName, channelName, message);
  124:             return res;
  125:
  126:     }
  127:
  128:     /**
  129:      * Calls DBSupport and returns the response
  130:      * @param senderName
  131:      * @param receiver
  132:      * @param message
```

```
  133:       * @return
  134:       */
  135:     public static DBSupport.HTTPResponse sendDirectMessage(String senderName
, String receiver, String message){
  136:         DBSupport.HTTPResponse res = DBSupport.sendDirectMessage(senderName,
 receiver, message);
  137:         return res;
  138:     }
  139: }
```

```
    1: package Models;
    2:
    3: import Controllers.DBSupport;
    4:
    5: /**
    6:  * Model for the channel, holds the same data that the server would and cont
ains the static calls to the dbsupport
    7:  */
    8: public class Channel {
    9:     private Integer id;
   10:
   11:     private Integer wId;
   12:
   13:     private String name;
   14:
   15:     /**
   16:      * Calls DBSupport and returns the response
   17:      * @param workspaceName
   18:      * @param name
   19:      * @return
   20:      */
   21:     public static DBSupport.HTTPResponse createChannel(String workspaceName,
 String name) {
   22:         DBSupport.HTTPResponse res = DBSupport.createChannel(workspaceName,
name);
   23:         return res;
   24:     }
   25:
   26:     /**
   27:      * Calls DBSupport and returns the response
   28:      * @param userName
   29:      * @param workspaceName
   30:      * @param channelName
   31:      * @return
   32:      */
   33:     public static DBSupport.HTTPResponse viewMentions(String userName, Strin
g workspaceName, String channelName) {
   34:         DBSupport.HTTPResponse res = DBSupport.viewMentions(userName, worksp
aceName, channelName);
   35:         return res;
   36:     }
   37:
   38:     /**
   39:      * Calls DBSupport and returns the response
   40:      * @param cId
   41:      * @return
   42:      */
   43:     public static DBSupport.HTTPResponse getChannelName(int cId) {
   44:         DBSupport.HTTPResponse res = DBSupport.getChannelName(cId);
   45:         return res;
   46:     }
   47:
   48:     public Integer getId() {
   49:         return id;
   50:     }
   51:
   52:     public void setId(Integer id) {
   53:         this.id = id;
   54:     }
   55:
   56:     public Integer getwId() {
   57:         return wId;
   58:     }
   59:
   60:     public void setwId(Integer wId) {
   61:         this.wId = wId;
   62:     }
   63:
   64:     public String getName() {
   65:         return name;
   66:     }
   67:
   68:     public void setName(String name) {
   69:         this.name = name;
   70:     }
   71:
   72:     public Channel(String name, int wID){
   73:         this.name = name;
   74:         this.id = -1;
   75:         this.wId = wID;
   76:     }
   77:
   78:     public Channel(String name, int wID, int id){
   79:         this.name = name;
   80:         this.id = id;
   81:         this.wId = wID;
   82:
   83:     }
   84:
   85:
   86: }
```

```
    1: package Models;
    2:
    3: import Controllers.DBSupport;
    4: /**
    5:  * Model for the user, holds the same data that the server would and contain
s the static calls to the dbsupport
    6:  */
    7: public class User {
    8:
    9:     private String name;
   10:     private String password;
   11:     private Integer userId;
   12:
   13:     public User(String uName, String uPassword){
   14:         name = uName;
   15:         password = uPassword;
   16:         userId = -1;
   17:     }
   18:
   19:     public User(String uName, String uPassword, Integer uId){
   20:         name = uName;
   21:         password = uPassword;
   22:         userId = uId;
   23:     }
   24:
   25:     /**
   26:      * Calls DBSupport and returns the response
   27:      * @param username
   28:      * @param password
   29:      * @return
   30:      */
   31:     public static DBSupport.HTTPResponse signIn(String username, String pass
word) {
   32:         DBSupport.HTTPResponse res = DBSupport.signin(username, password);
   33:         return res;
   34:     }
   35:
   36:     /**
   37:      * Calls DBSupport and returns the response
   38:      * @param senderId
   39:      * @return
   40:      */
   41:     public static DBSupport.HTTPResponse getUserNameByID(Integer senderId) {
   42:         DBSupport.HTTPResponse res = DBSupport.getUserNameByID(senderId);
   43:         return res;
   44:     }
   45:
   46:     public Integer getUserId() {
   47:         return userId;
   48:     }
   49:
   50:     public void setUserId(Integer userId) {
   51:         this.userId = userId;
   52:     }
   53:
   54:     public String getName() {
   55:         return name;
   56:     }
   57:
   58:     public void setName(String name) {
   59:         this.name = name;
   60:     }
   61:
   62:     public String getPassword() {
   63:         return password;
   64:     }
   65:
   66:     public void setPassword(String username) {
   67:         password = username;
   68:     }
   69:
   70:
   71:     public static DBSupport.HTTPResponse searchUser(String name){
   72:         DBSupport.HTTPResponse res = DBSupport.searchUser(name);
   73:         return res;
   74:     }
   75:
   76:     public static DBSupport.HTTPResponse createUser(String name, String pass
word){
   77:         DBSupport.HTTPResponse res = DBSupport.createUser(name, password);
   78:         return res;
   79:     }
   80:
   81: }
```

```
    1: package Models;
    2:
    3: import Controllers.DBSupport;
    4:
    5: /**
    6:  * Model for the Message Table. Essentially this is what the table will cont
ain
    7:  * @Author Dylan Mrzlak
    8:  */
    9: public class Textfile {
   10:
   11:     private Integer id;
   12:
   13:     private String name;
   14:
   15:     private String content;
   16:
   17:
   18:     public Message(Integer id,String name String content) {
   19:         this.id = id;
   20:         this.name = name;
   21:         this.content = content;
   22:      }
   23:
   24:     public Integer getId() {
   25:         return id;
   26:     }
   27:
   28:     public void setId(Integer id) {
   29:         this.id = id;
   30:     }
   31:     public String getName() {
   32:         return name;
   33:     }
   34:
   35:     public void setName(String content) {
   36:         this.name = name;
   37:     }
   38:
   39:     public String getContent() {
   40:         return content;
   41:     }
   42:
   43:     public void setContent(String content) {
   44:         this.content = content;
   45:     }
   46:
   47:     /**
   48:      * Calls DBSupport and returns the response
   49:      * @param message
   50:      * @return
   51:      */
   52:     public static DBSupport.HTTPResponse sendText(String name, String messag
e){
   53:         DBSupport.HTTPResponse res = DBSupport.sendMessage(name, message);
   54:         return res;
   55:
   56:     }
   57:
   58:     public static DBSupport.HTTPResponse getText(String name){
   59:         DBSupport.HTTPResponse res = DBSupport.sendMessage(name);
   60:         return res;
   61:
   62:     }
   63:
   64: }
```

```
    1: package Models;
    2:
    3: import Controllers.DBSupport;
    4: /**
    5:  * Model for the Workspace within the front end. Will contain the data and m
ethods required of the workspace
    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, w
e can only instantiate the name
   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:
   29:
   30:     public String getName(){
   31:         return name;
   32:     }
   33:
   34:     int getwId(){
   35:         return id;
   36:     }
   37:
   38:     public void setName(String name) {
   39:         this.name = name;
   40:     }
   41:
   42:     public int getId() {
   43:         return id;
   44:     }
   45:
   46:     public void setId(int id) {
   47:         this.id = id;
   48:     }
   49:
   50:     /**
   51:      * Create a workspace and call for the DBSUpport to request it put into
the DB
   52:      * @Author Dylan Mrzlak
   53:      */
   54:     public static DBSupport.HTTPResponse createWorkspace(String name){
   55:         DBSupport.HTTPResponse res = DBSupport.putWorkspace(name);
   56:         return res;
   57:     }
   58:
   59:     public static DBSupport.HTTPResponse searchWorkspace(String name){
   60:         DBSupport.HTTPResponse res = DBSupport.searchWorkspace(name);
   61:         return res;
   62:     }
   63:     /**
   64:      * Calls DBSupport and returns the response
   65:      * @param name
   66:      * @param username
   67:      * @return
   68:      */
   69:     public static DBSupport.HTTPResponse joinWorkspace(String name, String u
sername){
   70:         DBSupport.HTTPResponse res = DBSupport.joinWorkspace(name, username)
;
   71:         return res;
   72:     }
   73:
   74:     /**
   75:      * Calls DBSupport and returns the response
   76:      * @param mId
   77:      * @return
   78:      */
   79:     public static DBSupport.HTTPResponse pinMessage(String mId) {
   80:         DBSupport.HTTPResponse res = DBSupport.pinMessage(Integer.parseInt(m
Id));
   81:         return res;
   82:     }
   83:
   84:     /**
   85:      * Calls DBSupport and returns the response
   86:      * @param workspaceName
   87:      * @return
   88:      */
   89:     public static DBSupport.HTTPResponse getUsersInWorkspace(String workspac
eName){
   90:         DBSupport.HTTPResponse res = DBSupport.viewUsers(workspaceName);
   91:         return res;
   92:     }
   93: }
```

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

```
123:
124:     /**
125:      * Builds a request to create a user
126:      * @param name
127:      * @param password
128:      * @return
129:      */
130:     public static HTTPResponse createUser(String name, String password) {
131:         try {
132:             HTTPResponse response = serverRequest(ParamBuilder.createUser(na
me, password));
133:             return response;
134:         } catch (Exception e) {
135:             return new HTTPResponse(406, handleErr());
136:         }
137:     }
138:
139:     /**
140:      * Sets a message as pinned
141:      * @param id
142:      * @return
143:      * @Author Joseph Hudson
144:      */
145:     public static HTTPResponse pinMessage(Integer id) {
146:         try {
147:             HTTPResponse response = serverRequest(ParamBuilder.pinMessage(id
));
148:             return response;
149:         } catch (Exception e) {
150:             return new HTTPResponse(406, handleErr());
151:         }
152:     }
153:
154:     /**
155:      * Builds a request to send a message
156:      * @param senderName
157:      * @param workspaceName
158:      * @param channelName
159:      * @param message
160:      * @return
161:      */
162:     public static HTTPResponse sendMessage(String senderName, String workspa
ceName, String channelName, String message) {
163:         try {
164:             HTTPResponse response = serverRequest(ParamBuilder.sendMessage(s
enderName, workspaceName, channelName, message));
165:             return response;
166:         } catch (Exception e) {
167:             return new HTTPResponse(406, handleErr());
168:         }
169:     }
170:
171:     /**
172:      * Builds a request to create a channel
173:      * @param workspaceName
174:      * @param name
175:      * @return
176:      */
177:     public static HTTPResponse createChannel(String workspaceName, String na
me) {
178:         try {
179:             HTTPResponse response = serverRequest(ParamBuilder.addNewChannel
(workspaceName, name));
180:             return response;
181:         } catch (Exception e) {
182:             return new HTTPResponse(406, handleErr());
183:         }
184:     }
185:
186:     /**
187:      * Builds a request to send a DM
188:      * @param senderName
189:      * @param receiver
190:      * @param message
191:      * @return
192:      */
193:     public static HTTPResponse sendDirectMessage(String senderName, String r
eceiver, String message) {
194:         try {
195:             HTTPResponse response = serverRequest(ParamBuilder.sendDirectMes
sage(senderName, receiver, message));
196:             return response;
197:         } catch (Exception e) {
198:             return new HTTPResponse(406, handleErr());
199:         }
200:     }
201:
202:     /**
203:      * Builds a request to view users in a workspace
204:      * @param workspaceName
205:      * @return
206:      */
207:     public static HTTPResponse viewUsers(String workspaceName) {
208:         try {
209:             HTTPResponse response = serverRequest(ParamBuilder.getUsersInWor
kspace(workspaceName));
210:             return response;
211:         } catch (Exception e) {
212:             return new HTTPResponse(406, handleErr());
213:         }
214:     }
215:
216:     /**
217:      * Get all the mentions within a channel for a given user
218:      * @param username
219:      * @param workspaceName
220:      * @param channelName
221:      * @return Status code of the HTTP call and a response string (either a
JSON or a string)
222:      */
223:     public static HTTPResponse viewMentions(String username, String workspac
eName, String channelName) {
224:         try {
225:             HTTPResponse response = serverRequest(ParamBuilder.viewMentions(
username, workspaceName, channelName));
226:             return response;
227:         } catch (Exception e) {
228:             return new HTTPResponse(406, handleErr());
229:         }
230:     }
231:
232:     /**
233:      * Get all the messages within a workspace
234:      * @param workspaceName
235:      * @return Status code of the HTTP call and a response string (either a
JSON or a string)
236:      *          The Json is a list of messages all grouped by channel
237:      */
238:     public static HTTPResponse getAllMessages(String workspaceName) {
239:         try {
240:             HTTPResponse response = serverRequest(ParamBuilder.getAllMessage
s(workspaceName));
241:             return response;
242:         } catch (Exception e) {
243:             return new HTTPResponse(406, handleErr());
244:         }
245:     }
246:
```

```
247:     /**
248:      * searcher workspace
249:      * @param workspaceName
250:      * @return Status code of the HTTP call and a json list of the workspace
s
251:      */
252:     public static HTTPResponse searchWorkspace(String workspaceName) {
253:         try {
254:             HTTPResponse response = serverRequest(ParamBuilder.searchWorkspa
ce(workspaceName));
255:             return response;
256:         } catch (Exception e) {
257:             return new HTTPResponse(406, handleErr());
258:         }
259:     }
260:     /**
261:      * searcher workspace
262:      * @param workspaceName
263:      * @return Status code of the HTTP call and a json list of the workspace
s
264:      */
265:     public static HTTPResponse searchUser(String userName) {
266:         try {
267:             HTTPResponse response = serverRequest(ParamBuilder.searchUser(us
erName));
268:             return response;
269:         } catch (Exception e) {
270:             return new HTTPResponse(406, handleErr());
271:         }
272:     }
273:     /**
274:      * Builds a request to get a channel name
275:      * @param cId
276:      * @return
277:      */
278:     public static HTTPResponse getChannelName(int cId) {
279:         try {
280:             HTTPResponse response = serverRequest(ParamBuilder.getChannelNam
e(cId));
281:             return response;
282:         } catch (Exception e) {
283:             return new HTTPResponse(406, handleErr());
284:         }
285:     }
286:
287:     /**
288:      * Builds a request to get a user by id
289:      * @param senderId
290:      * @return
291:      */
292:     public static HTTPResponse getUserNameByID(Integer senderId){
293:         try {
294:             HTTPResponse response = serverRequest(ParamBuilder.getUserNameBy
Id(senderId));
295:             return response;
296:         } catch (Exception e) {
297:             return new HTTPResponse(406, handleErr());
298:         }
299:     }
300:
301:     /**
302:      * Builds a request to sign in a user
303:      * @param username
304:      * @param password
305:      * @return
306:      */
307:     public static HTTPResponse signin(String username, String password) {
308:         try {
309:             HTTPResponse response = serverRequest(ParamBuilder.signin(userna
me, password));
310:             return response;
311:         } catch (Exception e) {
312:             return new HTTPResponse(406, handleErr());
313:         }    }
314:
315:     /**
316:      * Model for the HTPPResponse rebuilding, that way the objects can handl
e the data themselve
317:      * @author Dylan Mrzlak
318:      */
319:     public static class HTTPResponse{
320:         public int code;
321:         public String response;
322:
323:         HTTPResponse(int status, String content){
324:             code = status;
325:             response = content;
326:         }
327:     }
328:
329:     /**
330:      * Static class to build our URL's to Strings.
331:      * Makes it a lot better to send it out to here, rather than build them
in other methods
332:      */
333:     private static class ParamBuilder{
334:         //This is the base url for our server. When we get a dedicated serve
r for the app, we will want this changed
335:         private static String BASE_URL = "http://localhost:8080/";
336:
337:         //Below are the Builders for the URL mappings
338:         //URLs are as follows:
339:         //      BASE_URL + CONTROLLER_MAPPING + / + REQUESTMAPPING + ?PARAM1
_NAME=PARAM1
340:         //For 2+ params:
341:         //      BASE_URL + CONTROLLER_MAPPING + / + REQUESTMAPPING + ?PARAM1
_NAME=PARAM1&PARAM2_NAME=PARAM2....
342:
343:         public static String sendDirectMessage(String sender, String recieve
r, String message){
344:             return BASE_URL+"/message/directMessage?senderName="+sender+"&re
cieverName="+reciever+"&message="+message;
345:         }
346:         public static String sendMessage(String sender, String workspace, St
ring channelName, String message){
347:             return BASE_URL+"/message/channelMessage?senderName="+sender+"&w
orkSpaceName="+workspace+"&channelName="+channelName+"&message="+message;
348:         }
349:
350:         public static String sendText(String name, String content){
351:             return BASE_URL+"/textfile/send?name="+name+"&content="+content;
352:         }
353:
354:         public static String getText(String name){
355:             return BASE_URL+"/textfile/download?name"+name;
356:         }
357:
358:         public static String createWorkspace(String name){
359:             return BASE_URL+"workspace/add?name="+name;
360:         }
361:
362:         public static String joinWorkspace(String workspaceName, String user
name){
363:             return BASE_URL+"user/join?workspaceName="+workspaceName+"&name=
"+username;
364:         }
365:
366:         public static String createUser(String name, String password){
```

```
  367:             return BASE_URL+"user/add?username="+name+"&password="+password;
  368:         }
  369:
  370:         public static String pinMessage(int mId){
  371:             return BASE_URL+"message/pinMessage?messageID=" + mId;
  372:         }
  373:
  374:         public static String getUsersInWorkspace(String workspaceName) {
  375:             return BASE_URL+"workspace/getUsers/?name="+workspaceName;
  376:         }
  377:         public static String searchWorkspace(String workspaceName) {
  378:             return BASE_URL+"workspace/search?name="+workspaceName;
  379:         }
  380:         public static String searchUser(String userName) {
  381:             return BASE_URL+"user/search?name="+userName;
  382:         }
  383:         public static String addNewChannel(String workspaceName, String name
) {
  384:             return BASE_URL+"channel/add?workspaceName="+workspaceName+"&nam
e="+name;
  385:         }
  386:
  387:         public static String viewMentions(String username, String workspaceN
ame, String channelName) {
  388:             return BASE_URL+"channel/viewMentions?username=" + username +
  389:                     "&workspaceName=" + workspaceName +
  390:                     "&channelName=" + channelName;
  391:         }
  392:
  393:         public static String getAllMessages(String workspaceName) {
  394:             return BASE_URL+"workspace/getAllMessages/?workspaceName="+works
paceName;
  395:         }
  396:
  397:         public static String getChannelName(int cId) {
  398:             return BASE_URL+"channel/getName?cId="+cId;
  399:         }
  400:
  401:         public static String getUserNameById(Integer senderId) {
  402:             return BASE_URL+"user/getUsername?senderId="+senderId;
  403:         }
  404:
  405:         public static String signin(String username, String password) {
  406:             return BASE_URL+"user/login?username="+username+"&password="+pas
sword;
  407:
  408:         }
  409:     }
  410: }
```

```
    1: import Controllers.DBSupport;
    2: import Models.Message;
    3: import Models.Channel;
    4: import Models.User;
    5: import Models.Workspace;
    6:
    7: import java.io.File;
    8: import java.io.FileWriter;
    9: import java.io.IOException;
   10: import java.text.DateFormat;
   11: import java.text.SimpleDateFormat;
   12: import java.util.Date;
   13: import java.util.Scanner;
   14:
   15: import com.google.gson.Gson;
   16:
   17:
   18: /**
   19:  * This will be the main controller for the application.
   20:  * It will take the initial input for User Input
   21:  * and then pass it along to other classes to handle the actual functionalit
y
   22:  *
   23:  * @Author Dylan Mrzlak
   24:  * Original Framework and Use handle for CREATE_WORKSPACE and JOIN_WORKSPACE
   25:  */
   26: public class InputController {
   27:     private static final String CREATE_WORKSPACE = "create workspace";
   28:     private static final String JOIN_WORKSPACE = "join";
   29:     private static final String CREATE_CHANNEL = "create channel";
   30:     private static final String VIEW_USERS = "view users";
   31:     private static final String SEND = "send";
   32:     private static final String SEND_DM = "send to";
   33:     private static final String ADD_USER = "create user";
   34:     private static final String PIN_MESSAGE = "pin message";
   35:     private static final String LOG_MESSAGES = "log messages";
   36:     private static final String VIEW_MENTIONS = "view mentions";
   37:     private static final String LOGIN = "login";
   38:     private static final String HELP = "help";
   39:     private static final String SEARCH_WORKSPACE = "search workspace";
   40:     private static final String SEARCH_USER = "search user";
   41:     private static final String SEND_TEXTFILE = "send textfile";
   42:     private static final String DOWNLOAD_TEXTFILE = "download textfile";
   43:
   44:
   45:     private static Gson gson = new Gson();
   46:     private static User curUser = null;
   47:     private static Workspace curWorkspace = null;
   48:     private static Channel curChannel = null;
   49:
   50:     public static void main(String[] args) {
   51:         //If this line get mad, check your dependencies, may have dropped
   52:         Scanner input = new Scanner(System.in);
   53:         String userInput = "";
   54:
   55:         printInstructions();
   56:         do {
   57:             userInput = input.nextLine();
   58:             //By forcing commands to be in a format of COMMAND - ARGUMENT
   59:             //We can easily manage the input and decide what is needed
   60:             int substringBegin = userInput.indexOf('-');
   61:             //Now that we have commands without args, we need to be able to
take commands without the delimitter set
   62:             if (substringBegin == -1) substringBegin = userInput.length();
   63:             String command = "";
   64:             String[] userArgs = {};
   65:             if (userInput.length() == substringBegin) {
   66:                 command = userInput;
   67:             } else {
   68:                 command = userInput.substring(0, substringBegin).trim();
   69:                 userArgs = userInput.substring(substringBegin + 1).trim().sp
lit(" ");
   70:             }
   71:             //Have updated the switch to be more readable and move into meth
ods, rather than holding all the logic in here.
   72:             //We were simply expanding this too much that it was becoming ha
rd to read. This is much more followable
   73:             switch (command) {
   74:                 case HELP:
   75:                     printHelp();
   76:                     break;
   77:                 case LOGIN:
   78:                     SignIn(userArgs);
   79:                     break;
   80:                 case ADD_USER:
   81:                     AddUser(userArgs);
   82:                     break;
   83:                 case CREATE_WORKSPACE:
   84:                     CreateWorkspace(userArgs);
   85:                     break;
   86:                 case JOIN_WORKSPACE:
   87:                     JoinWorkspace(userArgs);
   88:                     break;
   89:                 case CREATE_CHANNEL:
   90:                     CreateChannel(userArgs);
   91:                     break;
   92:                 case VIEW_USERS:
   93:                     ViewUsers(userArgs);
   94:                     break;
   95:                 case PIN_MESSAGE:
   96:                     PinMessage(userArgs);
   97:                     break;
   98:                 case SEND_DM:
   99:                     SendDM(userArgs);
  100:                     break;
  101:                 case SEND:
  102:                     SendMessage(userArgs);
  103:                     break;
  104:                 case LOG_MESSAGES:
  105:                     LogMessage(userArgs);
  106:                     break;
  107:                 case VIEW_MENTIONS:
  108:                     ViewMentions(userArgs);
  109:                     break;
  110:                 case SEARCH_USER:
  111:                     searchUser(userArgs);
  112:                     break;
  113:                 case SEARCH_WORKSPACE:
  114:                     searchWorkspace(userArgs);
  115:                     break;
  116:                 case DOWNLOAD_TEXTFILE:
  117:                     downloadTextfile(userArgs);
  118:                     break;
  119:                 case SEND_TEXTFILE:
  120:                     sendTextfile(usedArgs);
  121:                     break;
  122:                 default:
  123:                     System.out.println("Invalid Input please try again :(");
  124:                     break;
  125:             }
  126:         } while (input.hasNextLine());
  127:
  128:     }
  129:
  130:
  131:     private static void sendTextfile(){
  132:         if (curUser == null) {
  133:             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 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];

        DBSupport.HTTPResponse response = DBSupport.HTTPResponse Textfile.ge
tText(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 ig
nore 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 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;
        }
        //Now that those are out of the way, we need to get the actual menti
ons
        // depending on what the server returns, we handle it accordingly. W
e either get a list and print it,
        //  or an error and print that
        DBSupport.HTTPResponse response = Channel.viewMentions(curUser.getNa
me(), 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 base
d 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 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;
        }
        //Now that those are out of the way, we need to get the actual messa
ges.
        // 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.getNa
me());
        DBSupport.HTTPResponse response = Message.getAllMessages(curWorkspac
e.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.fo
rmat(date);
            System.out.println("Formatting");
            //We want to format the data as we want, and then take the new l
ist 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 cont
inue");
            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, w
orkspacesFound[].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 cont
inue");
            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 c
urrent 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 di
d 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 us
er 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;
```

```
373:         }
374:     }
375:
376:     /**
377:      * Create user in the server, then set the user to that
378:      * @param userArgs
379:      */
380:     private static void AddUser(String[] userArgs) {
381:         if (userArgs.length != 2) {
382:             System.out.println("Invalid Number or Arguments");
383:             return;
384:         }
385:         DBSupport.HTTPResponse uResponse = User.createUser(userArgs[0], user
Args[1]);
386:         if (uResponse.code > 300) {
387:             System.out.println(uResponse.response);
388:         } else {
389:             System.out.println("Saved User");
390:             User u = gson.fromJson(uResponse.response, User.class);
391:             curUser = u;
392:         }
393:     }
394:
395:     /**
396:      * Create a new workspace (if possible) and then set the user's current
workspace to that
397:      * @param userArgs
398:      * @author dylan mrzlak
399:      */
400:     private static void CreateWorkspace(String[] userArgs) {
401:         if (userArgs.length != 1) {
402:             System.out.println("Invalid Number or Arguments");
403:             return;
404:         }
405:         System.out.println("Creating Workspace...");
406:         DBSupport.HTTPResponse wResponse = Workspace.createWorkspace(userArg
s[0]);
407:         if (wResponse.code > 300) {
408:             System.out.println(wResponse.response);
409:         } else {
410:             System.out.println("Saved Workspace");
411:             Workspace w = gson.fromJson(wResponse.response, Workspace.class)
;
412:             curWorkspace = w;
413:             System.out.println("Joining Workspace");
414:             //Whenever a new workspace is created, the creator should automa
tically join it
415:             JoinWorkspace(new String[]{w.getName()});
416:             //We don't want to force a user to need to create a channel just
 to send messages, so now we automatically
417:             //make one, We can change the name whenever, but every new works
pace gets the same name for it's first channel
418:             System.out.println("Creating Default Channel");
419:             CreateChannel(new String[]{w.getName(), "Welcome"});
420:         }
421:     }
422:
423:     /**
424:      * Join a workspace (put data into the server that the user is in the wo
rkspace) and then make the workspace the current
425:      * @param userArgs
426:      * @author Dylan Mrzlak
427:      */
428:     private static void JoinWorkspace(String[] userArgs) {
429:         if (curUser == null) {
430:             System.out.println("You need to create a user or sign in to cont
inue");
431:             return;
432:         }
433:         if (userArgs.length != 1) {
434:             System.out.println("Invalid Number or Arguments");
435:             return;
436:         }
437:         System.out.println("Joining Workspace");
438:         DBSupport.HTTPResponse joinWorkspace = Workspace.joinWorkspace(userA
rgs[0], curUser.getName());
439:         if (joinWorkspace.code > 300) {
440:             System.out.println(joinWorkspace.response);
441:         } else {
442:             Workspace w = gson.fromJson(joinWorkspace.response, Workspace.cl
ass);
443:             curWorkspace = w;
444:             System.out.println("Joined Workspace " + w.getName() + " and set
 it to your current workspace");
445:         }
446:     }
447:
448:     /**
449:      * Create a new channel (if possible) and then set the user's current ch
annel to that
450:      * @param userArgs
451:      * @author dylan mrzlak
452:      */
453:     private static void CreateChannel(String[] userArgs) {
454:         if (curWorkspace == null) {
455:             System.out.println("User not in workspace");
456:             return;
457:         }
458:         if (userArgs.length != 2) {
459:             System.out.println("Wrong Number of arguments. Try: create chann
el - <workspace> <name> ");
460:             return;
461:         }
462:         DBSupport.HTTPResponse cResponse = Channel.createChannel(userArgs[0]
, userArgs[1]);
463:         if (cResponse.code > 300) {
464:             System.out.println(cResponse.response);
465:         } else {
466:             System.out.println("Saved Channel");
467:             Channel c = gson.fromJson(cResponse.response, Channel.class);
468:             curChannel = c;
469:         }
470:     }
471:
472:     /**
473:      * Get all of the users in a workspace, this is not a focused search at
all.
474:      * Simply a full list of users that are marked as having joined
475:      * @param userArgs
476:      * @author logan garrett
477:      */
478:     private static void ViewUsers(String[] userArgs) {
479:         if (curWorkspace == null) {
480:             System.out.println("User not in workspace");
481:             return;
482:         }
483:         //We either get an error and want to consume it, or we get a list to
 print
484:         DBSupport.HTTPResponse viewUsers = Workspace.getUsersInWorkspace(cur
Workspace.getName());
485:         if (viewUsers.code > 300) {
486:             System.out.println("There are no users in this workspace");
487:         } else {
488:             String[] userList = gson.fromJson(viewUsers.response, String[].c
lass);
489:             System.out.println("\nUsers in workspace: " + curWorkspace.getNa
me());
490:             for (int i = 0; i < userList.length; i++) {
```

```
491:                 System.out.println("\t" + userList[i]);
492:             }
493:             System.out.println(userList.length + " Users in this workspace f
ound. \n");
494:         }
495:     }
496:
497:     /**
498:      * Mark a message as pinned (when marked as pinned a pin search will be
able to get them)
499:      * A pinned message is technically important to the channel (but we're n
ot enforcing that and leaving that to the users)
500:      * @param userArgs
501:      * @author Joe Hudson
502:      */
503:     private static void PinMessage(String[] userArgs) {
504:         if (curUser == null) {
505:             System.out.println("You need to create a user or sign in to cont
inue");
506:             return;
507:         }
508:         if (userArgs.length != 1) {
509:             System.out.println("Invalid Number or Arguments");
510:             return;
511:         }
512:         DBSupport.HTTPResponse pinMessage = Workspace.pinMessage(userArgs[0]
);
513:         if (pinMessage.code > 300) {
514:             System.out.println(pinMessage.response);
515:         } else {
516:             System.out.println("Pinned message");
517:             Message m = gson.fromJson(pinMessage.response, Message.class);
518:             System.out.println("Message Pinned: \n\t" + "[" + m.getwId() + "
." + m.getcID() + "." + m.getId() + "]"
519:                     + m.getContent().replaceAll("_SS_", " "));
520:         }
521:     }
522:
523:
524:     /**
525:      * Send a message to the channel. Takes the content and will put it into
 the server.
526:      * @param userArgs
527:      * @author thomas mcandrew
528:      */
529:     private static void SendTextFile(String[] userArgs) {
530:         //null checks for the stuff that's required to send a message
531:         if (curUser == null) {
532:             System.out.println("You need to create a user or sign in to cont
inue");
533:             return;
534:         }
535:         if (curWorkspace == null) {
536:             System.out.println("User not in workspace");
537:             return;
538:         }
539:         if (curChannel == null) {
540:             System.out.println("User not in Channel;");
541:             return;
542:         }
543:         if (userArgs.length < 1) {
544:             System.out.println("Invalid number of arguments");
545:             return;
546:         }
547:         //Format the message in a way that the data can be sent fully, uncor
rupted
548:         //using _SS_ to replace 'spaces' in the message
549:         //We don't use http bodies, so the url is not a fan of spaces
550:         String message = "";
551:         for (int i = 0; i < userArgs.length; i++) {
552:             message += userArgs[i] + "_SS_";
553:         }
554:         message = message.trim();
555:         //Send the message to the server, and acknowledge the search
556:         DBSupport.HTTPResponse sendMessage = Message.sendMessage(curUser.get
Name(), curWorkspace.getName(), curChannel.getName(), message);
557:         if (sendMessage.code > 300) {
558:             System.out.println(sendMessage.response);
559:         } else {
560:             Message m = gson.fromJson(sendMessage.response, Message.class);
561:             System.out.println("Message Sent: \n\t" + m.getContent().replace
All("_SS_", " "));
562:         }
563:     }
564:
565:     /**
566:      * Send a message to the channel. Takes the content and will put it into
 the server.
567:      * @param userArgs
568:      * @author thomas mcandrew
569:      */
570:     private static void SendMessage(String[] userArgs) {
571:         //null checks for the stuff that's required to send a message
572:         if (curUser == null) {
573:             System.out.println("You need to create a user or sign in to cont
inue");
574:             return;
575:         }
576:         if (curWorkspace == null) {
577:             System.out.println("User not in workspace");
578:             return;
579:         }
580:         if (curChannel == null) {
581:             System.out.println("User not in Channel;");
582:             return;
583:         }
584:         if (userArgs.length < 1) {
585:             System.out.println("Invalid number of arguments");
586:             return;
587:         }
588:         //Format the message in a way that the data can be sent fully, uncor
rupted
589:         //using _SS_ to replace 'spaces' in the message
590:         //We don't use http bodies, so the url is not a fan of spaces
591:         String message = "";
592:         for (int i = 0; i < userArgs.length; i++) {
593:             message += userArgs[i] + "_SS_";
594:         }
595:         message = message.trim();
596:         //Send the message to the server, and acknowledge the search
597:         DBSupport.HTTPResponse sendMessage = Message.sendMessage(curUser.get
Name(), curWorkspace.getName(), curChannel.getName(), message);
598:         if (sendMessage.code > 300) {
599:             System.out.println(sendMessage.response);
600:         } else {
601:             Message m = gson.fromJson(sendMessage.response, Message.class);
602:             System.out.println("Message Sent: \n\t" + m.getContent().replace
All("_SS_", " "));
603:         }
604:     }
605:
606:
607:     /**
608:      * Send a message to a user. Takes the content and will put it into the
server.
609:      * DM's will be able to be seen by a user when a search for dm's is run
610:      * @param userArgs
611:      * @author thomas mcandrew
```

```
612:      */
613:     private static void SendDM(String[] userArgs) {
614:         if (userArgs.length < 2) {
615:             System.out.println("Invalid number of arguments");
616:             return;
617:         }
618:         //Format the message in a way that the data can be sent fully, uncor
rupted
619:         //using _SS_ to replace 'spaces' in the message
620:         //We don't use http bodies, so the url is not a fan of spaces
621:         String directMessage = "";
622:         for (int i = 1; i < userArgs.length; i++) {
623:             directMessage += userArgs[i] + "_SS_";
624:         }
625:         directMessage = directMessage.trim();
626:         DBSupport.HTTPResponse dm = Message.sendDirectMessage(curUser.getNam
e(), userArgs[0], directMessage);
627:         if (dm.code > 300) {
628:             System.out.println(dm.response);
629:         } else {
630:             System.out.println("Joining Workspace");
631:             Message m = gson.fromJson(dm.response, Message.class);
632:
633:             System.out.println("Message Sent: \n\t" + m.getContent().replace
All("_SS_", " "));
634:         }
635:     }
636:
637:     /**
638:      * Print the base instructions for the app, just a welcome to the app an
d a short description on how to operate it
639:      */
640:     private static void printInstructions() {
641:         System.out.println("Welcome to Slack# (patent pending), our cheeky,
user un-friendly, clone of Slack\n" +
642:                 "\t\tTo run this god forsaken app, type in a command and its
 arguments.\n" +
643:                 "\t\tIf you dont know the commands or need a refresher. I su
ggest you git gud skrub\n\n\n\n" +
644:                 "\t\t(Or enter \"help\", I'm not your mommy lol)");
645:     }
646:
647:     /**
648:      * Print the commands that have been implemented thus far
649:      */
650:     private static void printHelp() {
651:         System.out.println("Commands are sent in the order COMMAND - ARGUMEN
TS\n" +
652:                 "using ' ' to separate arguments\n\n" +
653:                 "create user: create user - <name> <password>\n" +
654:                 "login: login - <username> <password>\n" +
655:                 "create workspace: create workspace - <name of workspace>\n"
 +
656:                 "join workspace: join - <name of workspace>\n" +
657:                 "search workspace: search workspace - <name of workspace>\n"
 +
658:                 "create channel: create channel - <workspace name> <channel
name>\n" +
659:                 "view users: view users\n" +
660:                 "search user: search user - <name of user>\n"+
661:                 "send to group: send - <message>\n" +
662:                 "direct message: send to - <user> <message>\n" +
663:                 "pin message: pin message - <message>\n" +
664:                 "log messages: log messages\n" +
665:                 "view mentions: view mentions\n");
666:     }
667:
668:
669:
670:
671:
672:     private static void WriteFile(String[] linesToWrite, String filePath, St
ring fileName) {
673:         //Below is how we'll write to a file
674:         try {
675:             //We want to put it in the source directory of the entire projec
t so for Dylan (the author):
676:             //  "C:\Users\dmrz0\OneDrive\Desktop\Slack\logs\FILENAME"
677:             // Get that relative directory and if it doesn't exist. Make it
678:             File dir = new File(filepath);
679:             if(!dir.exists()){
680:                 dir.mkdir();
681:             }
682:             //Get the file for to write to.
683:             // It shouldn't really exist unless a user logs twice within a m
inute
684:             //If it does exist, delete it, and make a new one
685:             File toWrite = new File(dir + fileName + ".txt");
686:             FileWriter fw;
687:             if(toWrite.exists())
688:                 toWrite.delete();
689:             toWrite.createNewFile();
690:             //Set it to be writable
691:             toWrite.setWritable(true);
692:             //Prepare to start writing the file. Making a file Writer, and t
hen iteration through the data
693:             //and writing those lines into the file.
694:             fw = new FileWriter(toWrite);
695:             for(String line: linesToWrite){
696:                 fw.write(line);
697:             }
698:             //Close the writer to prevent memory leaks
699:             fw.close();
700:             //set the file to read only. Gotta keep our logs pure and clean
701:             toWrite.setReadOnly();
702:             System.out.println("File " + filePath + "Written to: \n" +
703:                     "Absolute Path: " + toWrite.getCanonicalPath() + "\n" +
704:                     "Relative Path: " + toWrite.getPath() + "\n");
705:         }
706:         catch (IOException e) {
707:             //Lots of methods have the chance to throw an error (although th
ey shouldn't now)
708:             //So we want to print that error.
709:             e.printStackTrace();
710:         }
711:     }
712:
713:     private static String[] LogMessagesFormat(Message[] messages) {
714:         String[] file = new String[messages.length];
715:         //We want to show the Workspace and Channel along with sender for ea
ch
716:         //As channel will change (and workspace will not) we want to keep tr
ack of the channel and get its name
717:         // when it changes. So we'll keep track of a messages cId.
718:         //We also want to have the Sender's name for each message, and that'
s not grouped,
719:         // so we'll need to pull that each message :(
720:         int cId = -1;
721:         String channelName = "";
722:         for (int i = 0; i < messages.length; i++) {
723:             String messageString = "";
724:             Message message = messages[i];
725:             if (message.getcID() != cId) {
726:                 cId = message.getcID();
727:                 DBSupport.HTTPResponse cRepsonse = Channel.getChannelName(cI
d);
728:                 if (cRepsonse.code >= 300) {
729:                     // as we want all messages that are public, should an is
```

```
sue from the backend happen,
  730:                     // we want to still display the message. What we'll do i
s make just use a tab for that
  731:                     channelName = "\t";
  732:                 } else {
  733:                     channelName = cRepsonse.response;
  734:                 }
  735:             }
  736:             String senderName;
  737:             DBSupport.HTTPResponse uRepsonse = User.getUserNameByID(message.
getSenderId());
  738:             if (uRepsonse.code >= 300) {
  739:                 // as we want all messages that are public, should an issue
from the backend happen,
  740:                 // we want to still display the message. What we'll do is ma
ke just use a tab for that
  741:                 senderName = "\t";
  742:             } else {
  743:                 senderName = uRepsonse.response;
  744:             }
  745:             messageString = "[" + curWorkspace.getName() + "].[" + channelNa
me + "]\t" + "FROM: " + senderName +
  746:                     "\n\tMESSAGE: " + message.getContent().replaceAll("_SS_"
, " ") + "\n";
  747:             file[i] = messageString;
  748:         }
  749:         return file;
  750:     }
  751: }
```

```
 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: }
```

```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);

```

```
124:         return new ResponseEntity(m, HttpStatus.OK);
125:
126:     }
127:
128: }
129:
```

```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 cont
ain
 * @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;
    }
}
```

```
    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:  * Interface for the given db table. Springboot will make all of the CRUD fu
nctions for us
    8:  * Anything past that that would require some kinda query, we need to put th
at SQL query here tied to a function
    9:  */
   10: public interface MessageRepository extends CrudRepository<Message, Integer>
{
   11:
   12:     @Query("SELECT CASE WHEN COUNT(m) > 0 THEN true ELSE false END FROM Mess
age m WHERE m.id = :id")
   13:     boolean existsByID(@Param("id") Integer id);
   14:
   15:     @Query("SELECT m FROM Message m WHERE m.id = :id")
   16:     Message findByID(@Param("id") Integer id);
   17:
   18:     @Query("Select m From Message m where m.wId = :wId AND m.cId = :cId")
   19:     Iterable<Message> getChannelMessages(@Param("wId") int wId, @Param("cId"
) int cId);
   20:
   21:     @Query("Select m From Message m where m.recipientID = :rId")
   22:     Iterable<Message> getUsersMessages(@Param("rId") int rId);
   23:
   24:
   25:     @Query("Select m From Message m where m.wId = :wId ORDER BY m.cId ASC, m
.id ASC")
   26:     Iterable<Message> getAllMessagesByWorkspace(@Param("wId") int wId);
   27:
   28:     @Query("Select m from Message m where m.wId = :wId AND m.cId = :cId AND
m.content LIKE :query")
   29:     Iterable<Message> getAllMessageContainsUName(@Param("query") String quer
y,
   30:                                                  @Param("wId") int wId,
   31:                                                  @Param("cId") int cId);
   32: }
```

```
    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: /**
   18:  * Interface for the given db table. Springboot will make all of the CRUD fu
nctions for us
   19:  * Anything past that that would require some kinda query, we need to put th
at SQL query here tied to a function
   20:  */
   21: public interface WorkspaceRepository extends CrudRepository<Workspace, Integ
er>{
   22:
   23:     @Query("SELECT CASE WHEN COUNT(w) > 0 THEN true ELSE false END FROM Work
space w WHERE w.name = :name")
   24:     boolean existsByName(@Param("name") String name);
   25:
   26:     @Query("SELECT w FROM Workspace w WHERE w.name = :name")
   27:     Workspace findbyName(@Param("name") String name);
   28:
   29:     @Query("SELECT w FROM Workspace w WHERE w.name LIKE :name")
   30:     Iterable<Workspace> searchWorkspace(@Param("name") String name);
   31: }
   32:
   33:
```

```
    1: package com.slack.server.workspace;
    2:
    3: import com.slack.server.messages.Message;
    4: import com.slack.server.messages.MessageRepository;
    5: import com.slack.server.user.User;
    6: import com.slack.server.user.UserRepository;
    7: import org.springframework.beans.factory.annotation.Autowired;
    8: import org.springframework.http.HttpStatus;
    9: import org.springframework.http.ResponseEntity;
   10: import org.springframework.stereotype.Controller;
   11: import org.springframework.web.bind.annotation.GetMapping;
   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: import java.util.HashMap;
   17:
   18: /**
   19:  * Controller for the Messages in the server
   20:  * We set a Mapping to a specified value, and all http requests that use tha
t
   21:  *      (BASE_URL + /mapping)
   22:  * Come here. This class handles all login for the given section
   23:  */
   24: @Controller    // This means that this class is a Controller
   25: @RequestMapping(path="/workspace") // This means URL's start with /demo (aft
er Application path)
   26: public class WorkspaceController {
   27:
   28:     /**
   29:      * Repo section
   30:      * Autowired gives the controller access to the specified repositories (
tables)
   31:      */
   32:     @Autowired
   33:     private WorkspaceRepository workspaceRepository;
   34:
   35:     @Autowired
   36:     private UserRepository userRepository;
   37:
   38:     @Autowired
   39:     private MessageRepository mRepo;
   40:
   41:     /**
   42:      * Create and put a Workspaceinto the table
   43:      * @param name
   44:      * @return
   45:      * @author Dylan Mrzlak
   46:      */
   47:     @GetMapping(path="/add") // Map ONLY POST Requests
   48:     public @ResponseBody ResponseEntity addNewWorkspace (@RequestParam Strin
g name) {
   49:         // @ResponseBody means the returned String is the response, not a vi
ew name
   50:         // @RequestParam means it is a parameter from the GET or POST reques
t
   51:         if(workspaceRepository.existsByName(name)) return new ResponseEntity
("Workspace name already taken", HttpStatus.NOT_ACCEPTABLE);
   52:         Workspace n = new Workspace();
   53:         n.setName(name);
   54:         workspaceRepository.save(n);
   55:         return new ResponseEntity(n, HttpStatus.OK);
   56:     }
   57:
   58:     /**
   59:      * Get all workspaces in DB
   60:      * @return
   61:      * @Author Dylan Mrzlak
   62:      */
   63:     @GetMapping(path="")
   64:     public @ResponseBody ResponseEntity getAllWorkspaces() {
   65:         // This returns a JSON or XML with the workspaces
   66:         Iterable<Workspace> list = workspaceRepository.findAll();
   67:         return new ResponseEntity(list, HttpStatus.OK);
   68:     }
   69:
   70:     /**
   71:      * Get a workspace by name from the DB
   72:      * @param name
   73:      * @return
   74:      * @Author Dylan Mrzlak
   75:      */
   76:     @GetMapping(path="/get")
   77:     public @ResponseBody ResponseEntity getWorkspaceByName(@RequestParam Str
ing name){
   78:         if(workspaceRepository.existsByName(name)) return new ResponseEntity
(workspaceRepository.findbyName(name), HttpStatus.OK);
   79:         return new ResponseEntity("Workspace does not exist", HttpStatus.NOT
_FOUND);
   80:     }
   81:
   82:     @GetMapping(path="/search")
   83:     public @ResponseBody ResponseEntity searchWorkspace(@RequestParam String
 name){
   84:         Iterable<Workspace> list;
   85:         if(name == "-1"){
   86:             list = workspaceRepository.findAll();
   87:         }else {
   88:             name = "%" + name + "%";
   89:             list =  workspaceRepository.searchWorkspace(name);
   90:         }
   91:         return new ResponseEntity(list, HttpStatus.OK);
   92:     }
   93:
   94:     /**
   95:      * Get all the users in a given workspace
   96:      * @param name
   97:      * @return
   98:      */
   99:     @GetMapping(path="/getUsers")
  100:     public @ResponseBody  ResponseEntity getUsersInWorkspace(@RequestParam S
tring name){
  101:         Iterable<String> list = userRepository.findUsers(name);
  102:         return new ResponseEntity(list, HttpStatus.OK);
  103:     }
  104:
  105:
  106:     @GetMapping(path="/getAllMessages")
  107:     public @ResponseBody ResponseEntity getAllMessages(String workspaceName)
 {
  108:         Workspace w = workspaceRepository.findbyName(workspaceName);
  109:         Iterable<Message> list = mRepo.getAllMessagesByWorkspace(w.getId());
  110:         return new ResponseEntity(list, HttpStatus.OK);
  111:     }
  112:
  113: }
```

```
    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.IDENTITY)
   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:  * Interface for the given db table. Springboot will make all of the CRUD fu
nctions for us
   13:  * Anything past that that would require some kinda query, we need to put th
at SQL query here tied to a function
   14:  */
   15: public interface UserRepository extends CrudRepository<User, Integer>{
   16:     @Query("SELECT CASE WHEN COUNT(u) > 0 THEN true ELSE false END FROM User
 u WHERE u.name = :name")
   17:     boolean existsByName(@Param("name") String name);
   18:
   19:     @Query("SELECT u FROM User u WHERE u.name = :name")
   20:     User findByName(@Param("name") String name);
   21:
   22:     @Query("SELECT u FROM User u WHERE u.id = :id")
   23:     User findByID(@Param("id") int id);
   24:
   25:     @Query("Select u.name "+
   26:             "From User u Left Join WorkspaceXRef x on u.id = x.uId "+
   27:             "where x.wId = (select id from Workspace w where w.name = :wName
)")
   28:     Iterable<String> findUsers(@Param("wName") String name);
   29:
   30:     @Query("SELECT u FROM User u WHERE u.name LIKE :name")
   31:     Iterable<User> searchUser(@Param("name") String name);
   32: }
```

```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);
  125:
  126:         //Chack that the user isn't already in the workspace
  127:         if(wXRefRepo.exists(w.getId(), u.getId())) return new ResponseEntity
("User already in Workspace", HttpStatus.NOT_ACCEPTABLE);
  128:
  129:         //Create the XREF and put it in DB
  130:         WorkspaceXRef x = new WorkspaceXRef();
  131:         x.setwId(w.getId());
  132:         x.setuId(u.getId());
  133:         wXRefRepo.save(x);
  134:
  135:         //Return OK status (200) and workspace
  136:         return new ResponseEntity(w, HttpStatus.OK);
  137:     }
  138:
  139:
  140:     /**
  141:      * Find a user by id, and return its username
  142:      * @param senderId
  143:      * @return
  144:      */
  145:     @GetMapping(path = "/getUsername")
  146:     public @ResponseBody ResponseEntity getUserNameById(Integer senderId) {
  147:         if(uRepo.existsById(senderId)) {
  148:             User user = uRepo.findByID(senderId);
  149:             return new ResponseEntity(user.getName(), HttpStatus.OK);
  150:         }
  151:         return new ResponseEntity("User Does Not Exist", HttpStatus.NOT_FOUN
D);
  152:
  153:     }
  154:
  155:
  156: }
```

```
    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: /**
    8:  * Interface for the given db table. Springboot will make all of the CRUD fu
nctions for us
    9:  * Anything past that that would require some kinda query, we need to put th
at SQL query here tied to a function
   10:  */
   11: public interface WorkspaceXRefRepository extends CrudRepository<WorkspaceXRe
f, Integer> {
   12:     @Query("SELECT CASE WHEN COUNT(x) > 0 THEN true ELSE false END FROM Work
spaceXRef x WHERE x.wId = :wId AND x.uId = :uId")
   13:     boolean exists(@Param("wId") int wId, @Param("uId") int uId);
   14: }
```

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

import javax.persistence.*;


/**
 * Model for the WorkspaceXRef Table. Essentially this is what the table wil
l 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 u
ser 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;
    }
}
```

```
    1: package com.slack.server.textfile;
    2:
    3: import com.slack.server.user.User;
    4: import org.springframework.data.jpa.repository.Query;
    5: import org.springframework.data.repository.CrudRepository;
    6: import org.springframework.data.repository.query.Param;
    7:
    8: public interface TextfileRepository extends CrudRepository<Textfile, Integer
> {
    9:     @Query("SELECT CASE WHEN COUNT(t) > 0 THEN true ELSE false END FROM Text
file t WHERE t.name = :name")
   10:     boolean existsByName(@Param("name") String name);
   11:
   12:     @Query("SELECT CASE WHEN COUNT(t) > 0 THEN true ELSE false END FROM Text
file t WHERE t.name = :name")
   13:     boolean existsById(@Param("name") int id);
   14:
   15:     @Query("SELECT t FROM Textfile t WHERE t.name = :name")
   16:     Textfile findByName(@Param("name") String name);
   17:
   18:     @Query("SELECT t FROM Textfile t WHERE t.id = :id")
   19:     Textfile findByID(@Param("id") int id);
   20: }
```

```
    1: package com.slack.server.textfile;
    2:
    3:
    4: import org.springframework.beans.factory.annotation.Autowired;
    5: import org.springframework.http.HttpStatus;
    6: import org.springframework.http.ResponseEntity;
    7: import org.springframework.stereotype.Controller;
    8: import org.springframework.web.bind.annotation.GetMapping;
    9: import org.springframework.web.bind.annotation.RequestMapping;
   10: import org.springframework.web.bind.annotation.RequestParam;
   11: import org.springframework.web.bind.annotation.ResponseBody;
   12:
   13: @Controller    // This means that this class is a Controller
   14: @RequestMapping(path="/user") // This means URL's start with /demo (after Ap
plication path)
   15: public class TextfileController {
   16:
   17:     @Autowired
   18:     private TextfileRepository tRepo;
   19:
   20:
   21:
   22:     @GetMapping(path="/send")
   23:     public @ResponseBody
   24:     ResponseEntity sendText(@RequestParam String name, @RequestParam String
content){
   25:         if(tRepo.existsByName(name)) return new ResponseEntity("Filename is
taken", HttpStatus.NOT_ACCEPTABLE);
   26:         Textfile t = new Textfile();
   27:         t.setName(name);
   28:         t.setContent(content);
   29:         tRepo.save(t);
   30:         return new ResponseEntity(t, HttpStatus.OK);
   31:     }
   32:     @GetMapping(path="/download")
   33:     public @ResponseBody
   34:     ResponseEntity sendText(@RequestParam String name){
   35:         if(!tRepo.existsByName(name)) return new ResponseEntity("File does n
ot exist", HttpStatus.NOT_FOUND);
   36:         Textfile t = tRepo.findByName(name);
   37:         return new ResponseEntity(t, HttpStatus.OK);
   38:     }
   39: }
```

```
 1: package com.slack.server.textfile;
 2:
 3: import org.springframework.data.annotation.Id;
 4:
 5: import javax.persistence.GeneratedValue;
 6: import javax.persistence.GenerationType;
 7:
 8: public class Textfile {
 9:     @Id
10:     @GeneratedValue(strategy= GenerationType.IDENTITY)
11:
12:     private Integer id;
13:
14:     private String name;
15:
16:     private String content;
17:
18:
19:     public Textfile(Integer id,String name, String content) {
20:         this.id = id;
21:         this.name = name;
22:         this.content = content;
23:     }
24:     public Textfile(String name, String content){
25:         this.id = -1;
26:         this.name = name;
27:         this.content = content;
28:     }
29:     public Textfile(){
30:         this.id = -1;
31:         this.name = null;
32:         this.content = null;
33:     }
34:
35:     public Integer getId() {
36:         return id;
37:     }
38:
39:     public void setId(Integer id) {
40:         this.id = id;
41:     }
42:     public String getName() {
43:         return content;
44:     }
45:
46:     public void setName(String content) {
47:         this.content = content;
48:     }
49:
50:     public String getContent() {
51:         return content;
52:     }
53:
54:     public void setContent(String content) {
55:         this.content = content;
56:     }
57:
58: }
```

```
    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:          * Base application to run the server. Springboot takes care of it f
or us, so we don't need much logic at all
   10:          * This class/method has visibility to the rest of the classes speci
fied and creates 'beans' for the repositories,
   11:          * creates the controllers onto the server page, and then the contro
llers handle logic
   12:          * @param args
   13:          */
   14:         public static void main(String[] args) {
   15:                 SpringApplication.run(ServerApplication.class, args);
   16:         }
   17:
   18: }
```

```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 user name 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) {
```

```
  122:         Channel c = channelRepository.findByID(cId);
  123:         if(c == null) return new ResponseEntity("Channel not found", HttpSta
tus.NOT_FOUND);
  124:         return new ResponseEntity(c.getName(), HttpStatus.OK);
  125:
  126:     }
  127: }
```

```
    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 cont
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.IDENTITY)
   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: /**
   10:  * Interface for the given db table. Springboot will make all of the CRUD fu
nctions for us
   11:  * Anything past that that would require some kinda query, we need to put th
at SQL query here tied to a function
   12:  */
   13: public interface ChannelRepository extends CrudRepository<Channel, Integer>{
   14:
   15:     @Query("SELECT CASE WHEN COUNT(c) > 0 THEN true ELSE false END FROM Chan
nel c WHERE c.wId = :wId AND c.name = :name")
   16:     boolean exists(@Param("wId") int wId, @Param("name") String name);
   17:
   18:     @Query("SELECT c FROM Channel c WHERE c.wId = :wId AND c.name = :name")
   19:     Channel find(@Param("wId") int wId, @Param("name") String name);
   20:
   21:
   22:     @Query("SELECT c FROM Channel c WHERE c.id = :cId")
   23:     Channel findByID( @Param("cId") int cId);
   24:
   25: }
   26:
   27:
```