```java
package ballmerpeak.turtlenet.server;

import ballmerpeak.turtlenet.client.Turtlenet;
import com.google.gwt.user.server.rpc.RemoteServiceServlet;
import java.io.*;
import java.security.*;
import ballmerpeak.turtlenet.server.TNClient;
import ballmerpeak.turtlenet.server.MessageFactory;
import ballmerpeak.turtlenet.shared.Message;
import ballmerpeak.turtlenet.shared.Conversation;
import ballmerpeak.turtlenet.shared.PostDetails;
import ballmerpeak.turtlenet.shared.CommentDetails;

@SuppressWarnings("serial")
public class TurtlenetImpl extends RemoteServiceServlet implements Turtlenet {
    TNClient c = null;

    public String startTN(String password) {
        Logger.init("LOG_turtlenet");
        Logger.write("INFO", "TNImpl","startTN(" + password + ")");
        c = new TNClient(password);
        if (c != null) {
            Thread t = new Thread(c);
            t.start();
            return "success";
        } else {
            return "failure";
        }
    }

    public String stopTN() {
        Logger.write("INFO", "TNImpl","stopTN()");
        c.running = false;
        return "success";
    }

    public String isFirstTime() {
        return !Database.DBExists() ? "true" : "false"; //GWT can only return objects
    }

    public String register(String username, String password) {
        Logger.init("LOG_turtlenet");
        Logger.write("INFO", "TnImpl", "Registering \"" + username + "\" with PW \"" + password + "\"");

        if (startTN(password).equals("success")) {
            while(!c.dbReady) {
                try{
                    Logger.write("CRAP", "TnImpl", "WAITING FOR DB");
                    Thread.sleep(1000);//TODO THIS IS AWFUL PRACTICE
                }catch(Exception e){}
            }

            Logger.write("INFO", "TnImpl", "Started TN...continuing registration");
            if (claimUsername(username).equals("success")) {
                addKey(Crypto.encodeKey(Crypto.getPublicKey()));
                return "success";
            } else {
                Logger.write("INFO", "TnImpl", "Username taken");
                Logger.write("INFO", "TnImpl", "---REGISTRATION FAIL#tUN---");
                return "taken";
            }
        } else {
            Logger.write("ERROR", "TnImpl", "Could not start Turtlenet");
            Logger.write("ERROR", "TnImpl", "---REGISTRATION FAIL#noTN---");
            return "failure";
        }
    }

    //Profile Data
    public String getMyUsername() {
        Logger.write("VERBOSE", "TnImpl", "getMyUsername()");
        return c.db.getName(Crypto.getPublicKey());
    }

    public String getUsername(String key) {
        Logger.write("VERBOSE", "TnImpl", "getUsername(" + key + ")");
        String name = c.db.getName(Crypto.decodeKey(key));
        Logger.write("VERBOSE", "TNImpl","getUsername returning \"" + name + "\"");
        return name;
    }

    public String getMyPDATA(String field) {
        Logger.write("VERBOSE", "TnImpl", "getMyPDATA(" + field + ")");
        return getPDATA(field, Crypto.encodeKey(Crypto.getPublicKey()));
    }

    public String getPDATA(String field, String key) {
        Logger.write("VERBOSE", "TnImpl", "getPDATA("+ field + ", ...)");
        return c.db.getPDATA(field, Crypto.decodeKey(key));
    }

    public String getMyKey() {
        Logger.write("VERBOSE", "TnImpl", "getMyKey()");
```

```
94          return Crypto.encodeKey(Crypto.getPublicKey());
95      }
96
97      public String getKey(String username) {
98          Logger.write("VERBOSE", "TnImpl", "getKey(" + username + ")");
99          return Crypto.encodeKey(c.db.getKey(username));
100     }
101
102     public String[][] getCategories () {
103         Logger.write("VERBOSE", "TnImpl", "getCategories()");
104         return c.db.getCategories();
105     }
106
107     public String[][] getPeople () {
108         Logger.write("VERBOSE", "TnImpl", "getPeople()");
109         return getCategoryMembers("all");
110     }
111
112     public Conversation[] getConversations () {
113         Logger.write("VERBOSE", "TnImpl", "START------------getConversations()");
114         Conversation[] conversations = c.db.getConversations();
115         for (int i = 0; i < conversations.length; i++) {
116             Logger.write("VERBOSE", "TnImpl", "\tSig: " + conversations[i].signature);
117             Logger.write("VERBOSE", "TnImpl", "\tTime: " + conversations[i].timestamp);
118             Logger.write("VERBOSE", "TnImpl", "\tFirst Message: " + conversations[i].firstMessage);
119             Logger.write("VERBOSE", "TnImpl", "\tUsers: " + conversations[i].users.length);
120             Logger.write("VERBOSE", "TnImpl", "\tKeys: " + conversations[i].keys.length);
121         }
122         Logger.write("VERBOSE", "TnImpl", "END  ------------getConversations()");
123         return conversations;
124     }
125
126     public Conversation getConversation (String sig) {
127         Logger.write("VERBOSE", "TnImpl", "getConversation(...)");
128         return c.db.getConversation(sig);
129     }
130
131     public String[][] getConversationMessages (String sig) {
132         Logger.write("VERBOSE", "TnImpl", "getConversationMessages(...)");
133         return c.db.getConversationMessages(sig);
134     }
135
136     public String[][] getCategoryMembers (String category) {
137         Logger.write("VERBOSE", "TnImpl", "getCategoryMembers(" + category + ")");
138         PublicKey[] keys = c.db.getCategoryMembers(category);
139         String[][] pairs = new String[keys.length][2];
140
141         for (int i = 0; i < keys.length; i++) {
142             pairs[i][0] = c.db.getName(keys[i]);
143             pairs[i][1] = Crypto.encodeKey(keys[i]);
144         }
145
146         return pairs;
147     }
148
149     public PostDetails[] getWallPosts (String key) {
150         Logger.write("VERBOSE", "TnImpl", "getWallPosts(...) ENTERING");
151         Message[] msgs = c.db.getWallPost(Crypto.decodeKey(key));
152         PostDetails[] posts = new PostDetails[msgs.length];
153         for (int i = 0; i < msgs.length; i++) {
154             String sig = msgs[i].getSig();
155             boolean liked = c.db.isLiked(sig);
156             int commentCount = c.db.getComments(sig).length;
157             Long time = msgs[i].getTimestamp();
158             String username = c.db.getName(Crypto.decodeKey(c.db.getWallPostSender(msgs[i].getSig())));
159             String text = msgs[i].POSTgetText();
160
161             posts[i] = new PostDetails(sig, liked, commentCount, time, username, text, Crypto.encodeKey(c.db.getSignatory(msgs
    [i])));
162         }
163         Logger.write("VERBOSE", "TnImpl", "getWallPosts(...) RETURNING");
164         return posts;
165     }
166
167     public CommentDetails[] getComments (String parent) {
168         Logger.write("VERBOSE", "TnImpl", "START----------getComments(...)");
169         Message[] commentMsgs = c.db.getComments(parent);
170         CommentDetails[] details = new CommentDetails[commentMsgs.length];
171
172         for (int i = 0; i < commentMsgs.length; i++) {
173             CommentDetails thisCmnt = new CommentDetails();
174             thisCmnt.posterKey = Crypto.encodeKey(c.db.getSignatory(commentMsgs[i]));
175             thisCmnt.posterName = c.db.getName(Crypto.decodeKey(thisCmnt.posterKey));
176             thisCmnt.sig = commentMsgs[i].getSig();
177             thisCmnt.text = commentMsgs[i].CMNTgetText();
178             thisCmnt.liked = c.db.isLiked(thisCmnt.sig);
179             details[i] = thisCmnt;
180         }
181         for (int i = 0; i < details.length; i++) {
182             Logger.write("VERBOSE", "TnImpl", "comment   sig: " + details[i].sig);
183             Logger.write("VERBOSE", "TnImpl", "comment  text: " + details[i].text);
184             Logger.write("VERBOSE", "TnImpl", "comment liked: " + details[i].liked);
185         }
```

```
186
187             Logger.write("VERBOSE", "TnImpl", "END -----------getComments(...)");
188             return details;
189         }
190
191         public Long timeMostRecentWallPost (String key) {
192             return c.db.timeMostRecentWallPost(Crypto.decodeKey(key));
193         }
194
195         public Long getConvoLastUpdated (String sig) {
196             String[][] details = c.db.getConversationMessages(sig);
197             if (details.length > 0)
198                 return Long.parseLong(details[details.length-1][1]);
199             else
200                 return 0L;
201         }
202
203         public Long getPostLastCommented (String sig) {
204             Message[] comments = c.db.getComments(sig);
205             return comments[comments.length-1].getTimestamp();
206         }
207
208         //Profile Data
209         public String claimUsername (String uname) {
210             Logger.write("VERBOSE", "TnImpl", "claimUsername(" + uname + ")");
211             c.db.addClaim(new MessageFactory().newCLAIM(uname));
212             if(c.connection.claimName(uname))
213                 return "success";
214             else
215                 return "failure";
216         }
217
218         public String updatePDATA (String field, String value) {
219             String ret = "success";
220             Logger.write("VERBOSE", "TnImpl", "updatePDATA(" + field + ", " + value + ")");
221             PublicKey[] keys = c.db.keysCanSeePDATA();
222             Message message = new MessageFactory().newPDATA(field, value);
223             for (int i = 0; i < keys.length; i++)
224                 if (!c.connection.postMessage(message, keys[i]))
225                     ret = "failure";
226             if (!c.connection.postMessage(message, Crypto.getPublicKey()))
227                 ret = "failure";
228             Parser.parse(message, c.db);
229             return ret;
230         }
231
232         public String updatePDATApermission (String category, boolean value) {
233             Logger.write("VERBOSE", "TnImpl", "updatePDATApermission(" + category + ", " + value + ")");
234             String ret = "success";
235
236             Message msg = new MessageFactory().newUPDATECAT(category, value);
237             ret = c.connection.postMessage(msg, Crypto.getPublicKey())?"success":"failure";
238             if (!c.db.updatePDATApermission(category, value))
239                 ret = "failure";
240             if (value) {
241                 PublicKey[] keys = c.db.getCategoryMembers(category);
242                 for (int i = 0; i < keys.length; i++) {
243                     if(!sendPDATA(Crypto.encodeKey(keys[i])).equals("success"))
244                         ret = "failure";
245                 }
246             }
247             Parser.parse(msg, c.db);
248
249             return ret;
250         }
251
252         //Posting
253         public String[] createCHAT (String[] keys) {
254             Logger.write("INFO", "TnImpl", "createCHAT(<" + keys.length + " keys>)");
255             String[] ret = new String[2];
256             ret[0] = "success";
257
258             String myStrKey = Crypto.encodeKey(Crypto.getPublicKey());
259             int count = 0;
260             int index = 0;
261             for (int i=0; i < keys.length; i++) {
262                 if (keys[i].equals(myStrKey)) {
263                     count++;
264                     index = i;
265                 }
266             }
267
268             //add self, or remove double self, from convo participants list
269             String[] newKeys = null;
270             if (count == 0) {
271                 newKeys = new String[keys.length+1];
272                 for (int i=0; i < keys.length; i++)
273                     newKeys[i] = keys[i];
274                 newKeys[keys.length] = myStrKey;
275                 keys = newKeys;
276             } else if (count == 2) {
277                 newKeys = new String[keys.length-1];
278                 int j = 0; //javac complains about `for (int i=0, int j=1;...' for some reason
```

```
279                 for (int i=0; i < keys.length; i++)
280                     if (i != index)
281                         newKeys[j++] = keys[i];
282                 keys = newKeys;
283             }
284 
285             Message msg = new MessageFactory().newCHAT(keys);
286             for (int i = 0; i < keys.length; i++)
287                 c.connection.postMessage(msg, Crypto.decodeKey(keys[i]));
288             Parser.parse(msg, c.db);
289 
290             Logger.write("VERBOSE", "TnImpl", "createCHAT returning " + msg.getSig());
291             ret[1] = msg.getSig();
292             return ret;
293         }
294 
295         public String addMessageToCHAT (String text, String sig) {
296             Logger.write("INFO", "TnImpl", "addMessageToCHAT(" + text + ",...)");
297             PublicKey[] keys = c.db.getPeopleInConvo(sig);
298             String ret = "success";
299 
300             if (keys.length == 0) {
301                 Logger.write("INFO", "TnImpl", "addMessageToCHAT(...) convo has " + Integer.toString(keys.length) + "
    participants");
302                 return "failure"; //Convo doesn't exist, or we don't know about it yet
303             }
304 
305             Logger.write("INFO", "TnImpl", "addMessageToCHAT(...) convo has " + Integer.toString(keys.length) + " participants");
306             Message msg = new MessageFactory().newPCHAT(sig, text);
307             for (int i = 0; i < keys.length; i++)
308                 if (!c.connection.postMessage(msg, keys[i]))
309                     ret = "failure";
310             Parser.parse(msg, c.db);
311             return ret;
312         }
313 
314         public String like (String sig) {
315             Logger.write("VERBOSE", "TnImpl", "like(...)");
316             PublicKey[] visibleTo = c.db.getVisibilityOfParent(sig);
317             Message message = new MessageFactory().newLIKE(sig);
318             String ret = "success";
319 
320             for (int i = 0; i < visibleTo.length; i++)
321                 if (!c.connection.postMessage(message, visibleTo[i]))
322                     ret = "failure";
323             if (!c.connection.postMessage(message, Crypto.getPublicKey()))
324                 ret = "failure";
325             Parser.parse(message, c.db);
326 
327             return ret;
328         }
329 
330         public String unlike (String sig) {
331             Logger.write("VERBOSE", "TnImpl", "unlike(...)");
332             PublicKey[] visibleTo = c.db.getVisibilityOfParent(sig);
333             Message message = new MessageFactory().newUNLIKE(sig);
334             String ret = "success";
335 
336             for (int i = 0; i < visibleTo.length; i++)
337                 if (!c.connection.postMessage(message, visibleTo[i]))
338                     ret = "failure";
339             if(!c.connection.postMessage(message, Crypto.getPublicKey()))
340                 ret = "failure";
341             Parser.parse(message, c.db);
342 
343             return ret;
344         }
345 
346         //Friends
347         public String addCategory (String name) {
348             Logger.write("VERBOSE", "TnImpl", "addCategory(" + name + ")");
349             Message msg = new MessageFactory().newADDCAT(name, false);
350 
351             return (c.db.addCategory(name, false) &&
352                     c.connection.postMessage(msg, Crypto.getPublicKey()))
353             ?"success":"failure";
354         }
355 
356         public String addToCategory (String group, String key) {
357             Logger.write("VERBOSE", "TnImpl", "addToCategory(" + group + ",...)");
358 
359             boolean alreadyMember = false;
360             PublicKey[] members = c.db.getCategoryMembers(group);
361             for (int i = 0; i < members.length; i++)
362                 if (members[i].equals(Crypto.decodeKey(key)))
363                     alreadyMember = true;
364 
365             if (!alreadyMember) {
366                 if (c.db.addToCategory(group, Crypto.decodeKey(key))) {
367                     Message msg = new MessageFactory().newADDTOCAT(group, key);
368                     c.connection.postMessage(msg, Crypto.getPublicKey());
369                     if (c.db.canSeePDATA(group)) {
370                         return sendPDATA(key).equals("success") ? "success" : "failure";
```

```
371                 } else {
372                     return "success";
373                 }
374 
375                 //We do not retroactivly send people posts/comments/likes because
376                 //  people will forget what they've posted in the past and accidently
377                 //  share it with new contacts.
378             } else {
379                 return "failure";
380             }
381         } else {
382             Logger.write("WARNING", "TnImpl", "Duplicate entry to tCategoryMembers prevented");
383             return "failure";
384         }
385     }
386 
387     public String sendPDATA (String key) {
388         String[] values = {"email", "name", "gender", "birthday"};
389         String[] fields = {getMyPDATA("email"), getMyPDATA("name"), getMyPDATA("gender"), getMyPDATA("birthday")};
390         return c.connection.postMessage(new MessageFactory().newPDATA(fields, values),
391                                         Crypto.decodeKey(key))
392                 ? "success" : "failure";
393     }
394 
395     public String removeFromCategory (String group, String key) {
396         Logger.write("VERBOSE", "TnImpl", "removeFromCategory(" + group + ",...)");
397         Message msg = new MessageFactory().newREMFROMCAT(group, key);
398         c.connection.postMessage(msg, Crypto.getPublicKey());
399         return c.db.removeFromCategory(group, Crypto.decodeKey(key))?"success":"failure";
400     }
401 
402     public String addKey (String key) {
403         Logger.write("VERBOSE", "TnImpl", "addKey(...)");
404         Message msg = new MessageFactory().newADDKEY(key);
405         return (c.db.addKey(Crypto.decodeKey(key)) &&
406                 c.connection.postMessage(msg, Crypto.getPublicKey())) ? "success":"failure";
407     }
408 
409     public String addPost (String wallKey, String categoryVisibleTo, String msg) {
410         Logger.write("VERBOSE", "TnImpl", "addPost(..., " + msg + ")");
411         PublicKey[] visibleTo = c.db.getCategoryMembers(categoryVisibleTo);
412         String[] visibleToStr = new String[visibleTo.length];
413         String ret = "success";
414 
415         for (int i = 0; i < visibleTo.length; i++)
416             visibleToStr[i] = Crypto.encodeKey(visibleTo[i]);
417         Message message = new MessageFactory().newPOST(msg, wallKey, visibleToStr);
418 
419         for (int i = 0; i < visibleTo.length; i++)
420             if (!c.connection.postMessage(message, visibleTo[i]))
421                 ret = "failure";
422         if (!c.connection.postMessage(message, Crypto.getPublicKey()))
423             ret = "failure";
424         Parser.parse(message, c.db);
425 
426         return ret;
427     }
428 
429     public String addComment (String parent, String text) {
430         Logger.write("VERBOSE", "TnImpl", "addComment(..., " + text + ")");
431         PublicKey[] visibleTo = c.db.getVisibilityOfParent(parent);
432         Message message = new MessageFactory().newCMNT(parent, text);
433         String ret = "success";
434 
435         Logger.write("VERBOSE", "TnImpl", "==================POSTING COMMENT TO " + visibleTo.length + " people");
436 
437         for (int i = 0; i < visibleTo.length; i++)
438             if (!c.connection.postMessage(message, visibleTo[i]))
439                 ret = "failure";
440         if(!c.connection.postMessage(message, Crypto.getPublicKey()))
441             ret = "failure";
442         Parser.parse(message, c.db);
443 
444         return ret;
445     }
446 
447     //Bad stuff
448     public String revokeMyKey () {
449         Logger.write("VERBOSE", "TnImpl", "-------revokeMyKey()-------");
450         PublicKey[] keys = c.db.getCategoryMembers("all");
451         String ret = "success";
452 
453         for (int i = 0; i < keys.length; i++)
454             if (!c.connection.postMessage(new MessageFactory().newREVOKE(0), keys[i])) //Can't be sent in cleartext,
    serverops could suppress it
455                 ret = "failure";
456 
457         //erase db and keypair
458         new File(Database.path + "/lastread").delete();
459         new File(Database.path + "/public.key").delete();
460         new File(Database.path + "/private.key").delete();
461         new File(Database.path + "/turtlenet.db").delete();
462         new File(Database.path).delete();
```

```
463
464             return ret;
465         }
466     }
```