```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()");
        return Crypto.encodeKey(Crypto.getPublicKey());
    }

    public String getKey(String username) {
        Logger.write("VERBOSE", "TnImpl", "getKey(" + username + ")");
        return Crypto.encodeKey(c.db.getKey(username));
    }

    public String[][] getCategories () {
        Logger.write("VERBOSE", "TnImpl", "getCategories()");
        return c.db.getCategories();
    }

    public String[][] getPeople () {
        Logger.write("VERBOSE", "TnImpl", "getPeople()");
        return getCategoryMembers("all");
    }

    public Conversation[] getConversations () {
        Logger.write("VERBOSE", "TnImpl", "START------------getConversations()");
        Conversation[] conversations = c.db.getConversations();
        for (int i = 0; i < conversations.length; i++) {
            Logger.write("VERBOSE", "TnImpl", "\tSig: " + conversations[i].signature);
            Logger.write("VERBOSE", "TnImpl", "\tTime: " + conversations[i].timestamp);
            Logger.write("VERBOSE", "TnImpl", "\tFirst Message: " + conversations[i].firstMessage);
            Logger.write("VERBOSE", "TnImpl", "\tUsers: " + conversations[i].users.length);
            Logger.write("VERBOSE", "TnImpl", "\tKeys: " + conversations[i].keys.length);
        }
        Logger.write("VERBOSE", "TnImpl", "END  ------------getConversations()");
        return conversations;
    }

    public Conversation getConversation (String sig) {
        Logger.write("VERBOSE", "TnImpl", "getConversation(...)");
        return c.db.getConversation(sig);
    }

    public String[][] getConversationMessages (String sig) {
        Logger.write("VERBOSE", "TnImpl", "getConversationMessages(...)");
        return c.db.getConversationMessages(sig);
    }

    public String[][] getCategoryMembers (String category) {
        Logger.write("VERBOSE", "TnImpl", "getCategoryMembers(" + category + ")");
        PublicKey[] keys = c.db.getCategoryMembers(category);
        String[][] pairs = new String[keys.length][2];

        for (int i = 0; i < keys.length; i++) {
            pairs[i][0] = c.db.getName(keys[i]);
            pairs[i][1] = Crypto.encodeKey(keys[i]);
        }

        return pairs;
    }

    public PostDetails[] getWallPosts (String key) {
        Logger.write("VERBOSE", "TnImpl", "getWallPosts(...) ENTERING");
        Message[] msgs = c.db.getWallPost(Crypto.decodeKey(key));
        PostDetails[] posts = new PostDetails[msgs.length];
        for (int i = 0; i < msgs.length; i++) {
            String sig = msgs[i].getSig();
            boolean liked = c.db.isLiked(sig);
            int commentCount = c.db.getComments(sig).length;
            Long time = msgs[i].getTimestamp();
            String username = c.db.getName(Crypto.decodeKey(c.db.getWallPostSender(msgs[i].getSig())));
            String text = msgs[i].POSTgetText();

            posts[i] = new PostDetails(sig, liked, commentCount, time, username, text, Crypto.encodeKey(c.db.getSignatory(msgs
[i])));
        }
        Logger.write("VERBOSE", "TnImpl", "getWallPosts(...) RETURNING");
        return posts;
    }

    public CommentDetails[] getComments (String parent) {
        Logger.write("VERBOSE", "TnImpl", "START----------getComments(...)");
        Message[] commentMsgs = c.db.getComments(parent);
        CommentDetails[] details = new CommentDetails[commentMsgs.length];

        for (int i = 0; i < commentMsgs.length; i++) {
            CommentDetails thisCmnt = new CommentDetails();
            thisCmnt.posterKey = Crypto.encodeKey(c.db.getSignatory(commentMsgs[i]));
            thisCmnt.posterName = c.db.getName(Crypto.decodeKey(thisCmnt.posterKey));
            thisCmnt.sig = commentMsgs[i].getSig();
            thisCmnt.text = commentMsgs[i].CMNTgetText();
            thisCmnt.liked = c.db.isLiked(thisCmnt.sig);
            details[i] = thisCmnt;
        }
        for (int i = 0; i < details.length; i++) {
            Logger.write("VERBOSE", "TnImpl", "comment   sig: " + details[i].sig);
            Logger.write("VERBOSE", "TnImpl", "comment  text: " + details[i].text);
            Logger.write("VERBOSE", "TnImpl", "comment liked: " + details[i].liked);
        }

        Logger.write("VERBOSE", "TnImpl", "END -----------getComments(...)");
        return details;
    }

    public Long timeMostRecentWallPost (String key) {
        return c.db.timeMostRecentWallPost(Crypto.decodeKey(key));
    }

    public Long getConvoLastUpdated (String sig) {
        String[][] details = c.db.getConversationMessages(sig);
        if (details.length > 0)
            return Long.parseLong(details[details.length-1][1]);
        else
            return 0L;
    }

    public Long getPostLastCommented (String sig) {
        Message[] comments = c.db.getComments(sig);
        return comments[comments.length-1].getTimestamp();
    }

    //Profile Data
    public String claimUsername (String uname) {
        Logger.write("VERBOSE", "TnImpl", "claimUsername(" + uname + ")");
        c.db.addClaim(new MessageFactory().newCLAIM(uname));
        if(c.connection.claimName(uname))
            return "success";
        else
            return "failure";
    }

    public String updatePDATA (String field, String value) {
        String ret = "success";
        Logger.write("VERBOSE", "TnImpl", "updatePDATA(" + field + ", " + value + ")");
        PublicKey[] keys = c.db.keysCanSeePDATA();
        Message message = new MessageFactory().newPDATA(field, value);
        for (int i = 0; i < keys.length; i++)
            if (!c.connection.postMessage(message, keys[i]))
                ret = "failure";
        if (!c.connection.postMessage(message, Crypto.getPublicKey()))
            ret = "failure";
        Parser.parse(message, c.db);
        return ret;
    }

    public String updatePDATApermission (String category, boolean value) {
        Logger.write("VERBOSE", "TnImpl", "updatePDATApermission(" + category + ", " + value + ")");
        String ret = "success";

        Message msg = new MessageFactory().newUPDATECAT(category, value);
        ret = c.connection.postMessage(msg, Crypto.getPublicKey())?"success":"failure";
        if (!c.db.updatePDATApermission(category, value))
            ret = "failure";
        if (value) {
            PublicKey[] keys = c.db.getCategoryMembers(category);
            for (int i = 0; i < keys.length; i++) {
                if(!sendPDATA(Crypto.encodeKey(keys[i])).equals("success"))
                    ret = "failure";
            }
        }
        Parser.parse(msg, c.db);

        return ret;
    }

    //Posting
    public String[] createCHAT (String[] keys) {
        Logger.write("INFO", "TnImpl", "createCHAT(<" + keys.length + " keys>)");
        String[] ret = new String[2];
        ret[0] = "success";

        String myStrKey = Crypto.encodeKey(Crypto.getPublicKey());
        int count = 0;
        int index = 0;
        for (int i=0; i < keys.length; i++) {
            if (keys[i].equals(myStrKey)) {
                count++;
                index = i;
            }
        }

        //add self, or remove double self, from convo participants list
        String[] newKeys = null;
        if (count == 0) {
            newKeys = new String[keys.length+1];
            for (int i=0; i < keys.length; i++)
                newKeys[i] = keys[i];
            newKeys[keys.length] = myStrKey;
            keys = newKeys;
        } else if (count == 2) {
            newKeys = new String[keys.length-1];
            int j = 0; //javac complains about `for (int i=0, int j=1;...' for some reason
            for (int i=0; i < keys.length; i++)
                if (i != index)
                    newKeys[j++] = keys[i];
            keys = newKeys;
        }

        Message msg = new MessageFactory().newCHAT(keys);
        for (int i = 0; i < keys.length; i++)
            c.connection.postMessage(msg, Crypto.decodeKey(keys[i]));
        Parser.parse(msg, c.db);

        Logger.write("VERBOSE", "TnImpl", "createCHAT returning " + msg.getSig());
        ret[1] = msg.getSig();
        return ret;
    }

    public String addMessageToCHAT (String text, String sig) {
        Logger.write("INFO", "TnImpl", "addMessageToCHAT(" + text + ",...)");
        PublicKey[] keys = c.db.getPeopleInConvo(sig);
        String ret = "success";

        if (keys.length == 0) {
            Logger.write("INFO", "TnImpl", "addMessageToCHAT(...) convo has " + Integer.toString(keys.length) + "
participants");
            return "failure"; //Convo doesn't exist, or we don't know about it yet
        }

        Logger.write("INFO", "TnImpl", "addMessageToCHAT(...) convo has " + Integer.toString(keys.length) + " participants");
        Message msg = new MessageFactory().newPCHAT(sig, text);
        for (int i = 0; i < keys.length; i++)
            if (!c.connection.postMessage(msg, keys[i]))
                ret = "failure";
        Parser.parse(msg, c.db);
        return ret;
    }

    public String like (String sig) {
        Logger.write("VERBOSE", "TnImpl", "like(...)");
        PublicKey[] visibleTo = c.db.getVisibilityOfParent(sig);
        Message message = new MessageFactory().newLIKE(sig);
        String ret = "success";

        for (int i = 0; i < visibleTo.length; i++)
            if (!c.connection.postMessage(message, visibleTo[i]))
                ret = "failure";
        if (!c.connection.postMessage(message, Crypto.getPublicKey()))
            ret = "failure";
        Parser.parse(message, c.db);

        return ret;
    }

    public String unlike (String sig) {
        Logger.write("VERBOSE", "TnImpl", "unlike(...)");
        PublicKey[] visibleTo = c.db.getVisibilityOfParent(sig);
        Message message = new MessageFactory().newUNLIKE(sig);
        String ret = "success";

        for (int i = 0; i < visibleTo.length; i++)
            if (!c.connection.postMessage(message, visibleTo[i]))
                ret = "failure";
        if(!c.connection.postMessage(message, Crypto.getPublicKey()))
            ret = "failure";
        Parser.parse(message, c.db);

        return ret;
    }

    //Friends
    public String addCategory (String name) {
        Logger.write("VERBOSE", "TnImpl", "addCategory(" + name + ")");
        Message msg = new MessageFactory().newADDCAT(name, false);

        return (c.db.addCategory(name, false) &&
                c.connection.postMessage(msg, Crypto.getPublicKey()))
        ?"success":"failure";
    }

    public String addToCategory (String group, String key) {
        Logger.write("VERBOSE", "TnImpl", "addToCategory(" + group + ",...)");

        boolean alreadyMember = false;
        PublicKey[] members = c.db.getCategoryMembers(group);
        for (int i = 0; i < members.length; i++)
            if (members[i].equals(Crypto.decodeKey(key)))
                alreadyMember = true;

        if (!alreadyMember) {
            if (c.db.addToCategory(group, Crypto.decodeKey(key))) {
                Message msg = new MessageFactory().newADDTOCAT(group, key);
                c.connection.postMessage(msg, Crypto.getPublicKey());
                if (c.db.canSeePDATA(group)) {
                    return sendPDATA(key).equals("success") ? "success" : "failure";
                } else {
                    return "success";
                }

                //We do not retroactivly send people posts/comments/likes because
                //  people will forget what they've posted in the past and accidently
                //  share it with new contacts.
            } else {
                return "failure";
            }
        } else {
            Logger.write("WARNING", "TnImpl", "Duplicate entry to tCategoryMembers prevented");
            return "failure";
        }
    }

    public String sendPDATA (String key) {
        String[] values = {"email", "name", "gender", "birthday"};
        String[] fields = {getMyPDATA("email"), getMyPDATA("name"), getMyPDATA("gender"), getMyPDATA("birthday")};
        return c.connection.postMessage(new MessageFactory().newPDATA(fields, values),
                                        Crypto.decodeKey(key))
               ? "success" : "failure";
    }

    public String removeFromCategory (String group, String key) {
        Logger.write("VERBOSE", "TnImpl", "removeFromCategory(" + group + ",...)");
        Message msg = new MessageFactory().newREMFROMCAT(group, key);
        c.connection.postMessage(msg, Crypto.getPublicKey());
        return c.db.removeFromCategory(group, Crypto.decodeKey(key))?"success":"failure";
    }

    public String addKey (String key) {
        Logger.write("VERBOSE", "TnImpl", "addKey(...)");
        Message msg = new MessageFactory().newADDKEY(key);
        return (c.db.addKey(Crypto.decodeKey(key)) &&
                c.connection.postMessage(msg, Crypto.getPublicKey())) ? "success":"failure";
    }

    public String addPost (String wallKey, String categoryVisibleTo, String msg) {
        Logger.write("VERBOSE", "TnImpl", "addPost(..., " + msg + ")");
        PublicKey[] visibleTo = c.db.getCategoryMembers(categoryVisibleTo);
        String[] visibleToStr = new String[visibleTo.length];
        String ret = "success";

        for (int i = 0; i < visibleTo.length; i++)
            visibleToStr[i] = Crypto.encodeKey(visibleTo[i]);
        Message message = new MessageFactory().newPOST(msg, wallKey, visibleToStr);

        for (int i = 0; i < visibleTo.length; i++)
            if (!c.connection.postMessage(message, visibleTo[i]))
                ret = "failure";
        if (!c.connection.postMessage(message, Crypto.getPublicKey()))
            ret = "failure";
        Parser.parse(message, c.db);

        return ret;
    }

    public String addComment (String parent, String text) {
        Logger.write("VERBOSE", "TnImpl", "addComment(..., " + text + ")");
        PublicKey[] visibleTo = c.db.getVisibilityOfParent(parent);
        Message message = new MessageFactory().newCMNT(parent, text);
        String ret = "success";

        Logger.write("VERBOSE", "TnImpl", "==================POSTING COMMENT TO " + visibleTo.length + " people");

        for (int i = 0; i < visibleTo.length; i++)
            if (!c.connection.postMessage(message, visibleTo[i]))
                ret = "failure";
        if(!c.connection.postMessage(message, Crypto.getPublicKey()))
            ret = "failure";
        Parser.parse(message, c.db);

        return ret;
    }

    //Bad stuff
    public String revokeMyKey () {
        Logger.write("VERBOSE", "TnImpl", "-------revokeMyKey()-------");
        PublicKey[] keys = c.db.getCategoryMembers("all");
        String ret = "success";

        for (int i = 0; i < keys.length; i++)
            if (!c.connection.postMessage(new MessageFactory().newREVOKE(0), keys[i])) //Can't be sent in cleartext,
serverops could suppress it
                ret = "failure";

        //erase db and keypair
        new File(Database.path + "/lastread").delete();
        new File(Database.path + "/public.key").delete();
        new File(Database.path + "/private.key").delete();
        new File(Database.path + "/turtlenet.db").delete();
        new File(Database.path).delete();

        return ret;
    }
}
```