

COMP208 - Group Software Project

Ballmer Peak

M. Chadwick; Choi, S.F (AKA Leon); P. Duff; L. Prince; A.Senin; L. Thomas

May 9, 2014

Contents

I Requirements	10
1 Mission Statement	11
2 Mission Objectives	12
3 Project Target	15
4 Threat Model	16
4.1 Scope	17
5 Anticipated Software	18
6 Anticipated Documentation	19
7 Anticipated Experiments and Their Evaluation	20
7.1 Performance Testing	20
7.2 Robustness Testing	20
7.3 Recoverability Testing	21
7.4 Learnability Testing	21
7.5 Security Testing	22
8 User View	23
8.1 System Boundary Diagram	24
9 User Requirements	26
9.1 Registration	26
9.2 Interacting with other users	26
9.3 Profile Data	27
9.4 Account recovery	27

CONTENTS	3
9.5 Posts	27
9.5.1 Walls	27
9.5.2 Commenting and Liking	28
9.5.3 Events	28
9.6 Chat	28
10 Case Study: Facebook	29
10.1 Overview	29
10.2 Registration	29
10.3 Account Management	30
10.4 Friend	30
10.5 Post	31
10.5.1 Posts, and functions thereof	31
10.5.2 Interaction with another's posts	32
10.6 Wall	32
10.7 Chat System	32
10.8 Architecture	33
10.9 Security	33
11 Case Study: Tor	34
11.1 Overview of Protocol	34
11.2 Security	34
12 Case Study: GPG and Email	36
13 Case Study: alt.anonymous.messages and Mix Networks	37
14 Security and Usability	38
14.1 Security and Usability	38
15 System Requirements	40
16 Transaction Requirements	42
16.1 Profile creation of the user	42
16.2 Adding of user relations	43
16.3 Assigning relations into categories	43
16.4 Adding of posts	43
16.5 Adding of events	43
16.6 User creating a new message	44

16.7 Receiving Content	44
17 Task List	45
18 Gantt Chart	48
19 Risk Assessment	54
19.1 Parallel Tasks	54
19.2 Group Work	54
19.3 Deadlines	55
19.4 Scope	55
19.5 Change Management	56
19.6 Stakeholders	56
19.7 Platforms	57
19.8 Integration	57
19.9 Requirements	57
19.10 Authority	58
19.11 External	58
19.12 Project Management	59
19.13 User Acceptance	59
19.14 Conclusion	60
II Design	61
20 Proposal Summary	62
21 Architecture	63
21.1 Network Architecture	63
21.2 System Architecture	64
21.3 Data Flow Diagram	64
22 Use Case Diagrams	66
23 Protocol	72
23.1 High Level Summary of Protocol	72
23.2 Client-Server Protocol	73
23.3 Client-Client Protocol	74
23.4 Summary	74
23.5 Message Formatting	74

CONTENTS	5
23.5.1 Unencrypted Messages	74
23.5.2 Encrypted Messages	75
23.6 Claiming a Username	75
23.7 Revoking a Key	75
23.8 Profile Data	75
23.9 Inter-User Realtime Chat	76
23.10 Posting to own wall	76
23.11 Posting on another users wall	77
23.12 Commenting	77
23.13 Liking	77
23.14 Events	77
24 Class Interfaces	79
24.1 Class Interfaces	79
24.2 Class Diagram	84
25 Pseudocode	85
25.1 Server	85
25.2 Client	85
25.3 Crypto	86
25.4 Database	87
25.5 Network Connection	88
25.6 Parser	89
26 Database	91
26.1 Database design description	91
26.1.1 user table	91
26.1.2 user, is_in_category, category table	91
26.1.3 user, is_invited, events table	91
26.1.4 user, allowed_to, wall_post table	92
26.1.5 user, has_like, wall_post table	92
26.1.6 user, has_like, has_comment table	92
26.1.7 user, has_comment, wall_post table	92
26.1.8 user, has_comment table	92
26.1.9 user, is_in_message, private_message table	93
26.1.10 message_claim table	93
26.1.11 key_revoke table	93
26.1.12 login_logout_log table	93

26.2 Logical table design version 1.0	94
27 Transaction details	97
28 User Interfaces	99
28.1 Interface Research	99
28.1.1 Swing	99
28.1.2 Abstract Window Toolkit	100
28.1.3 Standard Widget Toolkit	100
28.1.4 GWT	101
28.1.5 Javascript	101
28.2 GUI Design	102
28.2.1 Client Design	102
28.2.2 Server Design	104
28.3 Future Work	105
29 Business Rules	108
30 Data Dictionary	109
31 Glossary	115
III User Manual	117
IV Portfolio	140
32 Deviations in Requirements and Design	141
32.1 Logical table design version 2.0	142
32.2 Database design version 2.0	143
32.3 Expanded Client-Client Protocol	145
33 Turtlenet Website	146
34 Hosting	147
35 Source Code and Github.com	148

36 Testing	149
36.1 Automated Testing	149
36.2 Black-box Testing	149
36.3 Usability Testing	150
37 Future Development	151
37.1 Interface Framework	151
37.2 Interface and House Style	152
37.3 Languages Used	152
38 Ballmer Peak	154
38.1 Peter Duff - Team Lead	154
38.2 Luke Thomas - Lead Programmer	154
38.3 Louis Prince - Programmer	155
38.4 Aishah Senin - Database Developer	155
38.5 Michael Chadwick - Software Engineer	155
38.6 Leon Choi - Researcher	155
39 Professional Issues	156
40 Achievement of Objectives	157
41 Evaluation	158
Appendices	
A Minutes	161
A.1 Meeting #1 Minutes (Thursday, 30/01/2014)	161
A.2 Meeting #2 Minutes (Friday, 31/01/2014)	161
A.3 Meeting #3 Minutes (Tuesday, 04/02/2014)	161
A.4 Meeting #4 Minutes (Friday, 07/02/2014)	162
A.5 Meeting #5 Minutes (Wednesday, 12/02/2014)	162
A.6 Meeting #6 Minutes (Wednesday, 19/02/2014)	162
A.7 Meeting #7 Minutes (Friday, 07/03/2014)	163
A.8 Meeting #8 Minutes (Wednesday, 19/03/2014)	163
A.9 Meeting #9 Minutes (Friday, 21/03/2014)	165
A.10 Meeting #10 Minutes (Friday, 28/03/2014)	165
A.11 Meeting #11 Minutes (Sunday, 13/04/2014)	165
A.12 Meeting #12 Minutes (Saturday, 19/04/2014)	166

A.13 Meeting #13 Minutes (Friday 02/05/2014)	166
B Screenshots	168
C Deadlines	177
D TODO	178
D.1 General	178
D.2 Requirements Weeks 1-3	178
D.3 Design Weeks 4-X	179
E Bugs Accidental Features	181
F Licence	182
F.1 Statement of Purpose	182
F.2 Copyright and Related Rights	183
F.3 Waiver	183
F.4 Public License Fallback	184
F.5 Limitations and Disclaimers	184
F.6 Limitations places upon Turtlenet by The University of Liverpool	185
F.7 Included Works	194
F.8 The turtle image used as a favicon and in the banner	194
F.9 jquery.js Licence	194
F.10 todonotes.sty licence	195
F.11 The LaTeX Project Public License	195
F.11.1 PREAMBLE	195
F.11.2 DEFINITIONS	195
F.11.3 CONDITIONS ON DISTRIBUTION AND MODIFICATION	196
F.11.4 NO WARRANTY	198
F.11.5 MAINTENANCE OF THE WORK	199
F.11.6 WHETHER AND HOW TO DISTRIBUTE WORKS UNDER THIS LICENSE	200
F.12 CC0 licence and licence.png licence	202
F.13 Creative Commons Attribution 4.0 International Public License	202
F.13.1 Section 1 - Definitions.	203
F.13.2 Section 2 - Scope.	204
F.13.3 Section 3 - License Conditions.	205
F.13.4 Section 4 - Sui Generis Database Rights.	206
F.13.5 Section 5 - Disclaimer of Warranties and Limitation of Liability.	206
F.13.6 Section 6 - Term and Termination.	207

F.13.7 Section 7 - Other Terms and Conditions.	207
F.13.8 Section 8 - Interpretation.	207
F.14 ulem.sty licence	208
F.15 tikz-uml.sty licence	208
F.16 tikz-styles.sty licence	208
F.17 sqlite-jdbc-3.7.2.jar licence	208
F.17.1 1. Definitions.	208
F.17.2 2. Grant of Copyright License.	209
F.17.3 3. Grant of Patent License.	210
F.17.4 4. Redistribution.	210
F.17.5 5.	211
F.17.6 6.	211
F.17.7 7.	211
F.17.8 8.	211
F.17.9 9.	212
F.18 winstone.jar licence	212
F.19 CDDL licence licence	213
F.20 Unlicenced Material	213
G Source Listing	214
H Programmers Reference Manual	286
Todo list	344

Part I

Requirements

Chapter 1

Mission Statement

"to shift societal norms to a state wherein privacy is respected without caveat or justification"

In light of dissidents utilizing social networking websites such as Facebook and Twitter to organize protests, we feel that there is a need for an easy to use, encrypted communications platform with support for real-time and asynchronous communication between users.

Chapter 2

Mission Objectives

The proposed project (Turtlenet) is a simple, privacy oriented social network, which demands zero security or technical knowledge on behalf of its users. In order to ensure security and privacy in the face of nation state adversaries the system must be unable spy on its users even if it wants to or server operators intend to.

We feel that obscuring the content of messages isn't enough, because suspicion may, and often does, fall upon people not for what they say, but to whom they are speaking[24]. Our system will therefore not merely hide the content of messages, but the recipient of messages too. Hiding the fact that an IP address sent a message is out of scope, but hiding which user/keypair did so is in scope, as is which IP/user/keypair received the message and the content of the message. It is important to hide the recipient of the message, because otherwise they may be unfairly targeted[11] if they use our services to communicate with the wrong people on a phone which is later identified, or they may merely be 'selected' for spying and harassment [20, p. 3].

We feel that current tools have significant usability problems, as was recently made starkly clear when Glenn Greenwald, a reporter of the guardian, was unable to work with Edward Snowden because he found GPG to be "too annoying" to use.

"It's really annoying and complicated, the [email] encryption software" - Glenn Greenwald [20]

While there exist many tools for hiding what you are saying, relatively few seek to hide who talks with whom, and those which do often implement it merely as a proxy, or seek to provide convenience over security.

The system is to have strict security measures implemented. It is able to encrypt messages with the use of RSA and AES. The only way for the other user to decrypt the data is if it was encrypted using their public key; which is given from the recipient to the sender via whichever medium he

prefers, e.g. email. We will also allow users to transmit public keys as QR codes, for ease of use.

The system will provide a platform for people to securely communicate, both one-to-one and in groups. Users will be able to post information to all of their friends, or a subset of them as well as sharing links and discussing matters of interest.

The following are our main design goals. Please note that the system is designed with axiom that the server operators are unjust, seeking to spy on potential users, and able to modify the source for the server.

- Strong cryptography protecting the content of messages
- Make it an impossible task to derive, from the information the server has or is able to collect, which users send a message to which users
- Make it an impossible task to derive, from the information the server has or is able to collect, which users receive a message at all
- Transmission of public key is easy, and doesn't require knowing what a public key is
- Be intuitive and easy to use, prompting the user when required
- Provide a rich social network experience, so as to draw regular members and drive up network diversity

The server operator will have access to the following information:

- Which IP uploaded which message (although they will be ignorant of its content, type, recipient, and sender)
- Which IPs are connecting to the server as clients (but not what they view, whom they talk with, or whether they receive a message at all)
- What times a specific IP connects ¹

A third party logging all traffic between all clients and a server will have access to what IPs connect to the server, and whether they upload or download information ²

The benefits we feel this system provides over current solutions are:

- Server operators can not know who talks with whom
- Server operators can not know the content of messages
- Server operators can not know which message is intended for which user

¹While this will aid in tying an IP address to a person, it is deemed acceptable because it is not useful information unless the persons private key is compromised.

²size correlation attacks could be used here if the message content is known

- Server operators can not know who is friends with whom

In order to ensure nobody can tell who is talking with whom, we will base our security model on the idea of shared mailboxes, as seen in practice at alt.anonymous.messages³. In this model one posts a message by encrypting it using the public key of the recipient, and posting it in a public location. In this model, one reads a message by downloading all messages from that location, and attempting to decrypt them all using ones private key. Our protocol will build atop this simple premise, and the the server will be a mere repository of messages, the real work occurring wholly in the client.

³<https://groups.google.com/forum/#!forum/alt.anonymous.messages>

Chapter 3

Project Target

A project of this scope has a rather specific target in sight. Due to its encrypted nature, Turtlenet can act as a form of anonymity between users who would otherwise be targeted by governments and/or institutions opposed to them. Countries such as China[32] and a majority of the middle east[19] have recently seen negative press due to their persecution of individuals whom disagree with the ruling regime, such software would allow said individuals safety from what the wider world views as acceptable.

Large multinational defence corporations (e.g. IBM, Thales, BAE) might also find Turtlenet useful, as it would allow for a secure communication tool between employees in an office. It could also potentially be used outside a company firewall to send messages securely between offices across much larger distances. Corporations such as defence contractors often hold security in the highest regard, and such a project would match their needs well.

A more likely recipient of this system however, is society itself, as we have decided to waive our copyright granted monopoly. Should another group decide to embark on a similar project, they will have access to this project, to act as a baseline for their own work. See Appendix F.

Chapter 4

Threat Model

When designing a system in which security is a significant aspect, it helps to define clearly exactly what adversaries are anticipated. In this section we will describe a hypothetical adversary (hereafter 'the adversary') against whom we will protect our users.

The adversary will be granted all powers available to all conceivable attackers, such that no collusion of attackers may overcome our security (should it work for any given considered attacker).

The following individual attackers are considered, those attackers excluded are excluded on the basis that their abilities are a subset of the union of the abilities of the already considered attackers.

- Nation state without regard for international law and convention (e.g.: USA)
 - Pressure those it claims governance over into doing as it demands
 - Pressure companies operating within it into colluding in an attack
 - Identify all people connecting to the server. (Formed from the union of powers of the ISP and the server owner and operators)
- ISP (e.g.: BSkyB)
 - View all traffic on their network, after the point at which a user comes under suspicion.
 - Manipulate all traffic on their network however they desire.
 - Identify an IP address (during a specific time) with a person.
- Server Owners and Operators (i.e.: Those who own and operate Turtlenet)
 - Alter the source of the server in any way they desire.
 - Log all traffic before and after a user comes under attack.
 - Manipulate all traffic in any way they desire.

- Collect the IP of all connecting users.

Some of these claims may seem extreme, but given that companies such as BT, Vodafone Cable, Verizon, Level 3, and others have provided unlimited access to their networks[16] to governmental spy agencies, we feel it is a reasonable threat model in light of recent revelations[31].

Given that our system is intended to both protect people from the governments which claim governance over them, and mere greedy companies looking to sell or collect user data for profit, we will assume the worst case: i.e. that all our users, their ISPs, and the owners and operators of the Turtlenet server they use are able to be pressured by the adversary.

We grant the adversary all the powers listed above, and assume that all ISPs, companies, and Turtlenet server operators are actively working against all of our users. In summary, we consider the adversary to be:

A nation state for which money is no object, claims governance over the user, and has the ability to pressure service providers into spying on their users.

4.1 Scope

We do not attempt to protect against an adversary who has access to and the ability to modify the users hardware, nor do we attempt to conceal that an IP uploads data to the network.

While we recognise that the ability to post messages anonymously is important, especially considering that countries normally considered benign are prosecuting people over whom they claim governance for saying 'offensive' things [1], it is unfortunately out of scope for this project.

Chapter 5

Anticipated Software

We anticipate the creation of the following software:

- Windows, Linux, and OSX executable: client
- Windows, Linux, and OSX executable: server
- Windows, Linux, and OSX executable: installer for client and server
- Full source for server, client, and any associated works

The client will create and use an SQLite database, local to each client, this database will be used to store all information that the specific client is aware of.

Chapter 6

Anticipated Documentation

We will provide the following documentation:

- Installation guide for a server
- User manual for a client
- Full protocol documentation for third parties wishing to implement their own clients
- Full description of system design and architecture, for future maintenance
- Full description of database design
- Interface documentation

Chapter 7

Anticipated Experiments and Their Evaluation

7.1 Performance Testing

Evaluating how well the system performs under a high work load.

- Test to see how many simultaneous clients the server can handle.
- Test to see if the data received from the server under a high work load is accurate.
- Test the impact of a large number of clients on the servers response time.

A high work load will be simulated by automated clients performing user actions at random. The server should be capable of allowing these clients to communicate with one another quickly. The maximum number of concurrent clients possible without noticeable lag (twice the frequency of updates) should be recorded.

7.2 Robustness Testing

System level black box testing.

- Devise a series of inputs and expected outputs.
- Run these inputs through the system and record the actual outputs.
- Compare the actual outputs with the expected outputs.

- Simulate a denial of service attack. The server should be able to recover from the attack quickly and with minimal impact on the clients. Blocking such an attack is beyond the scope of this project.

Inputs used should range from expected use patterns to silly as users tend to do things totally unexpected. Expected and actual outputs should be recorded. Any differences will indicate problems with the system which need to be fixed.

7.3 Recoverability Testing

Evaluating how well the application recovers from crashes and errors.

- Restart the computer while the application is running. Ensure the local database is not corrupted.
- While the application is running terminate the computers network connection. Ensure the application continues working after the connection is re-established.
- Send a badly formatted message to another client. Ensure the application is able to keep running after receiving unexpected data from another client.

Each test should be run several times. If any test fails once or more this indicates that the system is bad at recovering from crashes and/or failures. In the case of a failure changes to the system should be implemented to improve recoverability.

7.4 Learnability Testing

Trialling the user interface with non expert users. Users should be able to use the system with minimal frustration and, ideally, without consulting the manual.

- Ensure users understand how to add friends, send messages, create posts, comment on posts and like posts.
- Ensure users don't spend excessive time searching for functions within the interface.
- Ensure error messages can be understood by the user and offer understandable advice on how to proceed.

Each test should be run several times with different users. If more than one user fails a test then changes need to be made to the interface. A single user experiencing problems is not an indication of a problem with the interface but instead suggests user incompetence.

7.5 Security Testing

The main goal of the system is to be secure. To ensure this goal is met the security of the system should be tested.

- Send non standard messages to clients. These should be rejected. If there is a flaw in the system the client may reveal information unintended for the recipient, in this case the program sending non standard messages.
- Recruit experienced programmers from outside of the group to attempt to penetrate or otherwise break the system. All attempts should be unsuccessful.

If any test fails this indicates a vulnerability in the system which should be corrected immediately. Security tests should be rerun after any changes during the testing phase to ensure new vulnerabilities are not introduced.

Chapter 8

User View

The user will be presented with a simple and easy to use interface, which assumes and requires no knowledge of security. The most complicated thing that the user will have to do is transmit to other users their public key. We plan to alleviate this process by encoding the public key as both a QR code and plaintext string (depending on user preference), both of which may be easily transmitted via email, SMS, meeting in person, or over any other channel.

Upon connecting to the system for the first time, the user will be prompted to enter a username, and any profile information they choose to share, and a passphrase. They will be urged to avoid using their real name as their username, and informed that profile information is shared on a case by case basis, and is not automatically visible to people whom they add. The entered passphrase will be used to deterministically derive an AES key which will be used to encrypt the users keypair and local DB. The user will be given the option of creating a second passphrase which, when entered, will overwrite the keypair and local DB with random data.

They will then be brought to the main page of the system, where they (and) people they authorize, may post message. There will be a prompt for them to add peoples public keys, and the option to add either a QR encoded or plaintext encoded public key.

Upon adding another's public key, they will first be informed of that persons username, and prompted to categorise the person. The user will be able to create a number of categories into which they can place that user. Already created categories will be displayed. One person may be added to multiple categories, and nobody but the user is aware that this occurs. Depending upon the categories the person is entered into, that person gains the ability to view certain content posted by the user.

When the user posts a message they are prompted to enter a recipient, this may be: a previously created category (such as friend, co-worker); a number of individuals; or any combination thereof.

Upon receiving a message a sound is played and the user is informed. They are then able to

click on the notification to open the message, and chat. When chatting with another user they have the ability to 'ignore' that user, in this case the user will see no more messages from that user.

8.1 System Boundary Diagram

Each client (of which there may be many) has his own client boundary consisting of his database and program client, whilst the server operators have their own boundary consisting of just the server. You can see in the diagram that at no point does the server operator or user functionality coincide with each other, leaving their privacy fully independent of one another. Each client (of which there may be many) has his own client boundary consisting of his database and program client, whilst the server operators have their own boundary consisting of just the server.

We can see the users interaction with the system below in the System Boundary Diagram:

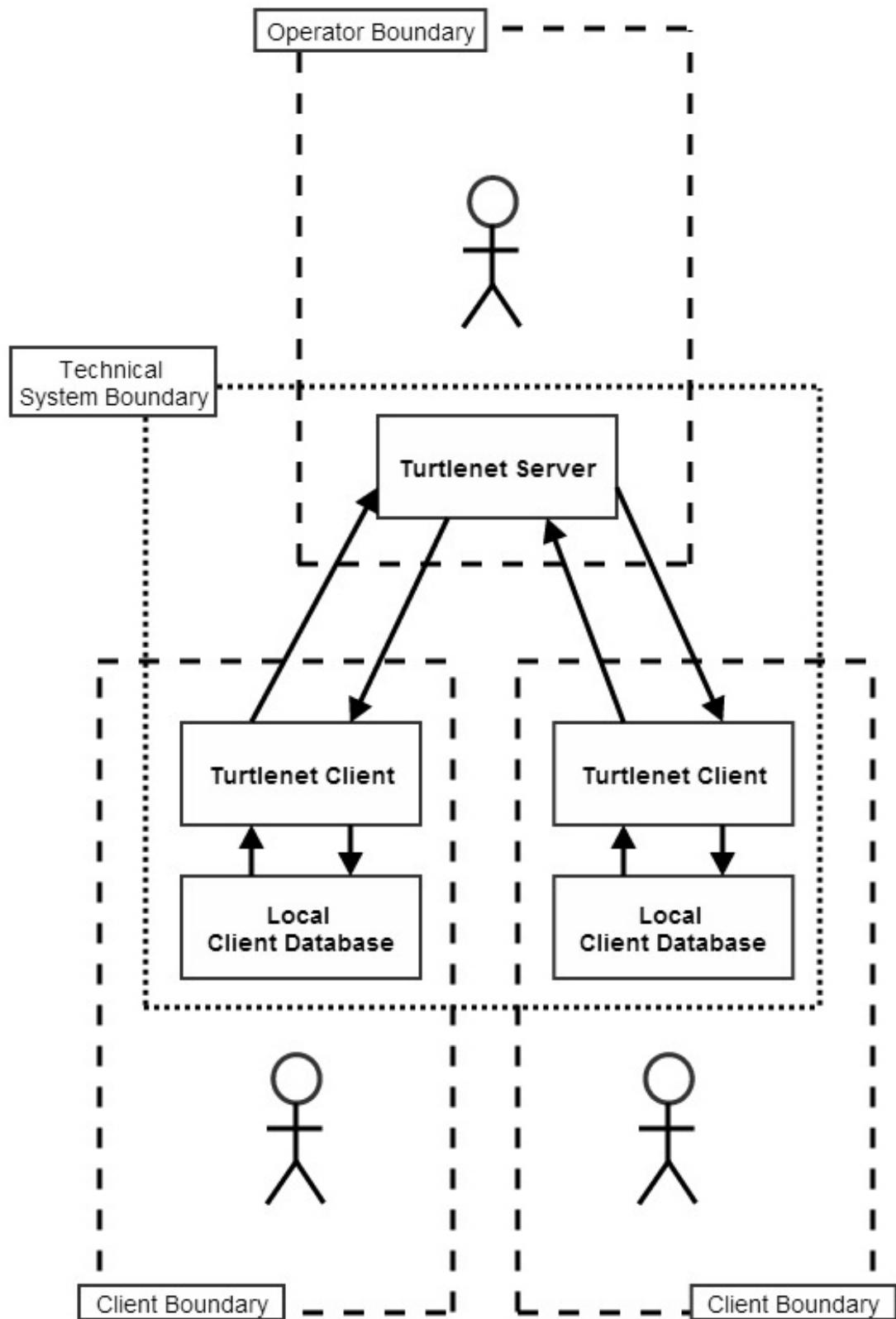


Figure 8.1: System Boundary Diagram

Chapter 9

User Requirements

9.1 Registration

Users may register by sending a CLAIM message to the server, this will claim a username for that user, and allow people they send messages to to see their username.

Before registering the user must generate an RSA keypair, they will be given the option of generating a new keypair, or using an existing keypair. The keypair provided will be encrypted using AES with the users password being used to derive the key. The user therefore must enter their password to log into the client. The database will be encrypted using the same AES key as the keys are encrypted with.

9.2 Interacting with other users

People are added by adding their public key, this is transmitted outside of our system, via whichever channel the users deem appropriate¹. We will provide a user with their public key as a QR code, or a plaintext string, depending on user preference.

Adding someone is asymmetric. Just because you add them doesn't mean they've added you. You do not require consent to add someone, just their public key.

The system allows the user to manage their list of known people into categories such as friends, family, and co-workers. The user defines these groups as lists of people whose public key they know. The user may create any group they desire, these groups are visible to only the user, and private.

¹This is required to prevent server operators from MitM'ing users

9.3 Profile Data

Profile data will be transmitted via PDATA messages. Different versions of profile information may be provided to different groups of people. Profile data may be updated by the user by future PDATA messages.

The supported fields in a PDATA message are:

- Name
- Username (unique, but this uniqueness is ensured by server and shouldn't be relied on)
- Birthday
- Sex
- E-Mail
- About
- Misc.

9.4 Account recovery

Account recovery is not possible without your keypair, due to this the GUI should urge the user to keep a copy on a flash drive, or external hard drive. The keys themselves will be encrypted with the users password.

9.5 Posts

9.5.1 Walls

Each user has their own wall. On their wall they may posts messages for themselves and others to see. All wall posts should be addressed to the user themselves so they can see their own posts, otherwise they will be unable to even view their own posts. When posting to their wall they choose who will be able to see the post, whether this is a group or people, a specific list of people, or just themselves is up to the user. They will not however be given the option to post publicly. Users may also post to another users wall.

Wall posts may contain links to other content, however this content is never thumb-nailed².

²client MUST NEVER thumbnail links or otherwise access external content without EXPLICIT user consent (see tor/js exploit on freedom hosting by the USA and tracking techniques recently thwarted by GMail now caching images. Specifically the fact that by delivering content over a secure channel that initiates communications outside of that channel, the recipients of content may be identified. A common variation of this is 'pixels' whereby a would be tracker embeds a 1x1 transparent PNG in a document, and watches who downloads that image from their servers.[28]

A user may edit their old posts, however older versions will still be available for viewing; similarly users may 'delete' posts, but they are still visible to malicious clients.

Due to bandwidth limitations on such networks as we are building, a user may only post plain-text, they may not post images, video, or audio.

9.5.2 Commenting and Liking

All wall posts may be commented on by any user who can see them. Comments are visible to all people who can see the original post; due to this, comments must be forwarded by the original posters client to all the same recipients, as the commenter may not know whom the original posters allowed to see the post.

Any wall post, comment, or other item on a wall may be liked.

9.5.3 Events

The client will alert the user to other users birthdays by automatically posting a wall post that only the user may read, which alerts the user of the event. These are otherwise normal wall posts. The user has the option of setting a category of people as a group for whom they desire to be alerted of events regarding.

Furthermore users may create their own events, for themselves and others to be alerted to. Recipients of events they did not create must accept the event before they are alerted of it.

9.6 Chat

Users may chat in real time, however messages can still be sent when one user logs off, to be received when they log in. Past conversations are saved, and a user may block users from messaging them; the client actually just ignores their messages, it's impossible to stop someone from messaging you.

Conversations may include two or more people.

Chapter 10

Case Study: Facebook

10.1 Overview

A user has a profile with information about them, they may add other users as 'friends', friends may view each others 'posts' and talk to each other. Posts are multimedia messages typically visible to all the friends of the person who made the post. Most posts can be commented upon, and both posts and comments may be 'liked'. Liking merely publicly marks the fact that you approve of something.

10.2 Registration

In order to be a user of facebook, one must register. In doing so you provide facebook with the following information, this may also be used to later reset the password of your account, should you forget it.

- First Name
- Last Name
- E-Mail
- Password
- Birthday
- Sex

In order to register one must read and agree to their terms [10], read their data use policy [9], and read their cookie policy [8]. Given profile information can be changed at a later date, within certain bounds. Facebook requires the use of your real name, and in fact forbids all false personal information, under their terms.[10, p. 4.1]

10.3 Account Management

The user is given the ability to set the security defaults for their posts and information. These options include who is able to see wall posts, whether comments are enabled by default, and who may see which aspects of your profile information. You can also manage the permissions granted to facebook apps.

Access may be gained to an account by knowing certain information, the intent is to allow people to recover their account if they forget their password.

A users profile may contain the following information:

- Work and education
- Place of Birth
- Relationship
- Basic Information
 - Birthday
 - Relationship
 - Status
 - Anniversary
 - Languages
 - Religious
 - Political
 - Family
 - Contact Information

10.4 Friend

In facebook, 'friending' someone is symmetric; that is, if you are friends with them, they are friends with you. The facebook servers store which user is friends with which other users. Adding another

Field	Description
Photo	All the photos the user's has tagged
Friend	What friends the user has
Note	What notes the users up/downloaded to facebook
Groups	What groups the user has join
Events	What events user may be attending
Likes	What page(s) (unknown type) the user liked
Apps	What apps the user has
Books	What book pages the user liked/followed
TV programmes	What TV pages the user liked/followed
Films	What films pages the user liked/followed
Music	What music(or stars) the user liked/followed
Sports	What sport pages the user liked
Place	Where's the user has been

Table 10.1: The user adds a new post

user as a friend is simply a matter of sending that user a friend request, and having it approved by the second user. A user may see a list of all who are their 'friend' on FB, in the friend list. After friending somebody that persons wall posts will appear on your news feed, and you will be able to chat with that user.

In order to add friends, facebook allows you to see your friends friend lists, and search by name, email, and location for other users. Facebook also suggests other users whom you may already know IRL, based on your friends friends. Non-users are also able to search facebook for people that they may know.

10.5 Post

10.5.1 Posts, and functions thereof

Facebook allows a user to post on their wall or friend's wall (if they are friends with the facebook user). Posts may contain: text, images, videos, or any combination thereof.

A user posting a post may do the following:

- Delete their own post
- Rewrite their own post
- Decide who may view a post, the options are as follows:
 - Public
 - Private

- Only-me
- Friends only
- Friends of friends

10.5.2 Interaction with another's posts

A post will typically be displayed on the news feeds of the people who are able to see it, due to this the name of the person who made a post is always displayed next to it. Posts themselves may be commented upon, liked, and reposted to the viewers wall ('shared') with an additional message; the number and names of people who have liked a post is displayed underneath it; likes may be cancelled at a later date. The comment function however, may be disabled by the user who makes the post.

A user may hide specific posts, or hide all posts by a specific user. They may also, instead of hiding another's posts all together, merely prevent them from being automatically displayed on their news feed. A user may report an image, video or comment to facebook team (e.g. the post is offensive). Comments may also be liked, hidden, and reported; following such a report FB is able to remove offensive or illegal posts.

Images which are posted may be tagged, this allows other users to mouse-over parts of the image and be informed who is pictured. This functionality is also used to add all posted images of someone to their profile.

10.6 Wall

A users wall stores all the posts of the user posted since the account was created and the information about the user, this information is presented in reverse chronological order, so that recent events are at the top of the page and easily visible. Other users may view the users wall by clicking the name of the user from anywhere in facebook. Other users may post on a friends wall along with it's owner (see section on posts for more information); in this case, both the poster and the owner of the wall can delete the post. Facebook also retains the power to erase any content on its service.

Posts mentioning a user are automatically reposted to that users wall, this can occur manually or when that person is tagged in an image.

10.7 Chat System

Facebook allows a user to chat with their friends, and will inform a user of whether their friends are online or not (though this can be faked), and whether the user you are chatting with has read

the last message that you sent them. You are also informed whether your friend is logged in on a mobile device or not.

Whole groups of users may chat together, in multi-user conversations. Facebook also supports video calling and file transfer during chats. If a user does not wish to be bothered by another using chatting with them, then they may 'mute' that users conversion. Users spamming via chat may be reported to facebook. Because multi-user conversations (and indeed long running one-to-one conversations) can get rather large, facebook allows you to hide the history of a conversation.

Facebook chat alerts the user to new messages in a conversation by playing a sound.

10.8 Architecture

From a users point of view facebook is ostensibly organised as a single central server; we are here concerned with the general architecture and not the specific implementation of it, and so we will consider all of facebooks servers to be a single server for the purposes of this section.

Users connect to facebook using a web browser, and proceed to download a client written in javascript. User data is uploaded to facebook over HTTP as cleartext. The data is stored on unencrypted on facebooks servers, and facebook maintains a database of all data.

This allows clients to download only the data they need, as they can simply ask for it. This in turn means that facebooks current architecture can, and does, support a huge user-base, measured in the millions.

10.9 Security

In order to use facebook after registration a user must 'log in'. This places an authentication cookie on the users computer which gives anyone in possession of it the ability to act as that user.

If the user logs in from an IP associated with a region geographically far from the last login, facebook will confirm that the user owns the account by asking them to identify a friend in a photograph, or by other means.

Facebook chat turns the users computer into a server, whereby facebooks central server sends messages to the client as it receives them, rather than the client requesting new messages. This has been used in the past to identify facebook users by correlating sent messages of specific size sent at a specific time.

Facebook has access to all its users data, and is able to erase, modify, and fabricate it. Facebook is aware of everything which happens on facebook. Censorship is a common occurrence on facebook.

Chapter 11

Case Study: Tor

11.1 Overview of Protocol

Tor is an implementation of onion routing, it routes traffic from your computer through a number of other nodes; the final 'exit' node routes the traffic to the final destination[15]. Node IP's are listed publicly in directory servers. In this manner the IP of clients connecting to a server is obscured from that server.

RSA/AES is used to ensure that only you, the exit node, and the final destination see the plaintext traffic being routed. With the use of TLS, SSL, or other end-to-end encryption those who see the plaintext can be reduced to you and the final destination. However a malicious exit node can MitM SSL connections using ssl-strip or a similar tool. There are methods of avoiding this, but it is a serious issue because users believe that SSL is secure. This exploit is found in the wild[18], and so is most definitely a concern.

Tor also supports 'hidden services' which seek to conceal the IP of the client from the server, and the IP of the server from the client. These are significantly more secure as the traffic never exits the tor network, however provide no protection from the adversary as will be described later; after all, we're assuming the server operators are colluding, so they will provide data required for traffic confirmation.

11.2 Security

Given that Tor is a low-latency network, traffic can easily be correlated. This problem is ameliorated in high-latency networks such as mix nets, but not eliminated.

Tor does not seek to protect against size correlation, or time correlation of traffic. Rather the purview of tor is to conceal the IP address of a client from the servers which it connects to.

Should a global passive adversary have perfect visibility of the internet, they would be able to track tor traffic from source to host by correlating the size and time of transmissions.

The Tor design doesn't try to protect against an attacker who can see or measure both traffic going into the Tor network and also traffic coming out of the Tor network[7]. -
Roger Dingledine

We can safely assume that the adversary has access to the clients traffic, since our threat model is that of a nation state seeking to spy on its citizens. Furthermore we may assume that the adversary has access to the content host, as our threat model assumes that service operators may be pressured legally or otherwise into spying on their users. Therefore we must conclude, at least for *the* adversary, that Tor is unsuitable for concealing activity in traditional social networks, due to traffic confirmation.

Does this then mean that Tor is insecure? No. So far as we know[30] the US does not currently have the ability to reliably and consistently track tor users; if the US is incapable of doing so, it is reasonable to assume that no other nation state has this ability. This is however not something which should be relied upon, as assumptions widely lead to mistakes. We shall therefore consider Tor as unsuitable for transmitting our data, at least if we were to do so as a traditional social network.

With manual analysis we can de-anonymize a very small fraction of Tor users, however, no success de-anonymizing a user in response to a TOPI request/on demand[30].

Chapter 12

Case Study: GPG and Email

GPG is an implementation of the PGP[2], providing both public/private key encryption and also a number of symmetric ciphers that can be used separately.

It is common practice to use GPG to encrypt email, and several popular addons for browsers exist to aid in this[12]. Unfortunately GPG itself is difficult to use[20], and a significant barrier to entry.

The encrypting of email with RSA¹ is a good solution if one wants to keep the content of messages secure, and unmodified. However it is out of scope for PGP to hide who is communicating, so while we find the underlying cryptography sound, our scope is simply too different for PGP to be of any use; with one exception.

Public key distribution is a significant challenge². PGP partially solves this problem by introducing the concept of a 'web of trust'. In such a system one marks public keys as trusted, presumably the keys of people you trust, and the people whom you have marked as trusted can then sign the keys of other people whom they trust. These keys may then be distributed, with the RSA signatures of everyone who signed them, to everyone. If I download a key and see that it has been verified by someone that I trust, then I can trust that key (albeit less than the original key). This in combination with the small world hypothesis³[29] allows a large number of public keys to become known to a user merely by adding one friend's key, and having the client automatically sign all keys it comes across from a trusted source.

We will take the 'web of trust' into consideration during design, however it may present some significant security issues.

¹and a symmetric cipher

²Our system can't do it, or it would be trivial to MitM users who don't check they received the correct key via another channel

³The phenomenon that people in the earth's population seem to be separated by at most 6 intermediaries.

Chapter 13

Case Study: alt.anonymous.messages and Mix Networks

alt.anonymous.messages is a newsgroup¹ to which people publicly post encrypted messages. In order to retrieve messages a recipient downloads all new messages and attempts to decrypt them all, those which they are able to decrypt are read, and others ignored.

This type of system is known as a 'shared mailbox', and is often not used by hand, but by mix network servers, which provide high-latency email forwarding, and handle the encryption on behalf of the users. Mix-networks massively slow timing-based traffic confirmation because they cache a large number of messages before sending them all out at once in a random order[26].

This system provides the property we are seeking: concealing who talks with whom on our network, even from the server itself. This property is ensured by the fact that the server cannot tell who reads a specific message, even though it knows which IP uploaded it. It also introduces a huge amount of overhead, in the form of downloading everyone else's messages as well as ones own.

Mix networks however have some serious issues, and misconfiguration easily allows for traffic correlation[25], albeit not confirmation (without a large sample size). Furthermore mix networks only function if the operator is trusted, this is unacceptable against our threat model. For these reasons we will not use the idea of mix networks.

We have identified the method of operation of shared mailboxes as the basis for our communications protocol, and will build a social network on top of this concept.

¹It may be accessed, without an installed Usenet client, through several websites providing an interface to it, one such website is Google Groups, and may be accessed here: <https://groups.google.com/forum/#!forum/alt.anonymous.messages>

Chapter 14

Security and Usability

14.1 Security and Usability

There is room for improvement when it comes to increasing the level of usability and user-friendliness in the Turtlenet software.

Firstly, as developers, we need to pay attention to the target audience, especially when it alludes to the general public, which have different ranges of user experience and knowledge when using a social network program. Paying attention and understand to how the general public thinks and the level of their knowledge are, we as developers can shape the program into something that is user-friendly and avoid mistakes which can be easily made by the users if the program is confusing. We have to be aware that since Turtlenet is a social media which is more security oriented, there are some functionalities that the general public might not be knowledgeable about.

In order to help the users who uses Turtlenet for the first time in our future development, is the use of 'pop up manual'. These instructions will pop out in a subtly manner in specific areas on the website when the user uses Turtlenet for the first time. An example is under the public key field. The general public would have no idea what a public key is or what is to do with it. A pop up manual is useful for this case, it will pop out saying in a generic, non-jargon or technical sense saying "Public key is the key to share with your trusted friends! Hence its name, it behaves like a key which can be passed along to others so they can unlock and view your profile."

Lastly, in order to make future improvements of the program, one important and effective way to do so is to listen to the customers' feedback. There are several platforms to gather feedback from customers, one common way to do it nowadays is using social media websites, and Twitter will be the best option for this case. Users can easily mention us in tweets, could be anything from actual feedback, rants, questions regarding the use of the program, what they like about it, and positive feedback. This could become useful to us developers to see how the customer think, feel and behave

towards Turtlenet, and possibly the trends about their use of it. Using such information, developers can improve on certain areas of the program according to the users needs. Users are the one who uses the program regularly and know what is best when it comes to functionality.

Chapter 15

System Requirements

An estimate is hereafter given as to the size of all stored messages, and the amount of data which would need downloading by each client when it is started. The following assumptions are used throughout:

- A users average message posted to their wall is 200 characters
- A users average number of messages posted to their wall per day is 10
- A users average number of friends is 100 (each and every friend represents one key exchange)
- A users average private message (to single user) is 50 characters
- A users average number of private (to single user) messages per day is 300

With these generous estimates, each user would generate $(200*10*100)+(50*300*1)$ bytes of raw data per day. Assuming a 10% protocol overhead we would see 236,500 bytes of data per day per user.

The storage space required for a server is therefore 86MB per year per user. On a server with 50,000 users that has been running for 3 years, there would be just 1.3TB of data.

Every time a client connects, it must download all messages posted since it last connected to the server. To mitigate this we may run as a daemon on linux, or a background process in windows, that starts when the user logs in. If we can expect a computer to be turned on for just 4 hours a day then 20 hours of data must be downloaded. $((236,500*\text{no_of_users})/24)*\text{hours_off_per_day}$ bytes must be downloaded when the users computer is turned on.

The following table shows the delays between the computer turning on, and every message having been downloaded (assuming a download speed of 500KB/second, and a network of 1000 users).

Hours off per day	Minutes to sync
0	0
4	1.3
10	3.2
12	3.9
16	5.2
20	6.5

Table 15.1: Hours a computer is turned off per day vs minutes to sync

We feel that waiting 2-5 minutes is an acceptable delay for the degree of privacy provided. Once the user is synced after turning their computer on, no further delays will be incurred until the computer is shut down.

Due to the inherently limited network size (<1500 users of one server is practical) we recommend a number of smaller servers, each serving either a geographic location, or a specific interest group.

While this latency could be avoided, and huge networks ($>1,000,000$) used, it would come at the cost of the server operator being able to learn that somebody is sending or receiving messages, and also who those messages are sent to/from (although they couldn't know what the messages said).

The server therefore merely needs a fast internet connection to upload and download content from clients. The client is required to perform a significant amount of encryption and decryption, however the client will almost certainly be able to encrypt/decrypt faster than a connection to the internet so the network speed may be considered the limiting factor for users on the internet [5]. Large companies however may very well use the system over a LAN, however these can be reasonably expected to have fairly modern computers which can more than handle RSA decryption.

Chapter 16

Transaction Requirements

Due to the nature of Turtlenet there may exist no central database, rather each client maintains their own local database of everything they know. The data forming this is all stored centrally, however to build a complete database would require the private key of every user of the service, which clearly we do not have access to.

There are 3 categories of data transaction:

- 1. Data entry
- 2. Data update and deletion
- 3. Data queries

16.1 Profile creation of the user

All that is required to use a Turtlenet server is a valid RSA keypair. Users don't have accounts per se, but rather associate profile data with a public key if they so desire. Users have no login information, rather posts are authenticated via RSA signatures. Usernames are the sole public information in our system, and as such each client has a complete list of usernames.

When a client first connects it is advisable, albeit not required, to claim a username. This is done merely by posting that username, and a signed hash of it to the server. Therefore the DB must store all such CLAIM messages.

Optional profile data which the user may enter is stored as PDATA messages, and the database will be required to store these.

16.2 Adding of user relations

Communication between people on Turtlenet requires that one is in possesion of the public key of the recipient, and should they wish to respond then they must be in possesion of your public key. We define 'A being related to B' to mean that A is in possesion of B's public key, and B is in possesion of A's public key. This is given a special name as it is a very common situation.

A user may be uniquely identified by their public key, and it may be used to derive their username, if they claimed one. Being in a relation with someone doesn't mean that you can see any profile information of theirs, however the GUI will ask the user whether they wish to share their own profile information with someone when they add that persons key.

16.3 Assigning relations into categories

When a user adds a relation, he has a choice of adding him into a specific category (or categories). A user can create any category he wants by going to the options and click 'Add new category'. The database then records the new category into the category table. The user then can then assign the relation into the existing category.

Categories are useful because they allow the user to share their posts with a predetermined set of people automatically, withing having to list each individual as a recipient.

16.4 Adding of posts

Users may post on their, or - with permission - others, walls. A post has a list of people who can see it (the user may choose a previously defined category or a specific list of people) however this list isn't public so only the DB of the author of a post (and the owner of the wall) will contain information as to who is able to see it.

The post itself has a timestamp, a signature (authenticating it), and content. The databse will store all of these.

NB: When a user posts something, they are automatically added to the list of recipients. A users own posts are downloaded from the server, just like everyone elses, and are in no way special.

16.5 Adding of events

The database will store events, these may be created by the user, or recieived from other users. At the appropriate time the GUI will notify the user of an event occuring. Example include birthdays, deadlines, and important dates. Events recieived from other users must be accepted by the user

before the GUI will alert the user of them, for this reason the DB must also store whether an event is accepted or not.

16.6 User creating a new message

A user can initiate a conversation with (an)other user(s) by creating a new message. Messages are merely a special case of wall posts, which are handled differently by the GUI.

16.7 Receiving Content

When the client connects it will download all messages posted since it last connected, it will then attempt to decrypt them all using the users private key. Those messages which are successfully decrypted are authenticated by verifying the signature and the content added to the database. It is in this manner that all content is passed from server to client.

Chapter 17

Task List

Task ID	Task Description (Desc.)	Due Date	Deliverable
1	Project Planning	14/02/2014	Planning segment
1.1	Mission Statement	07/02/2014	Same as Desc.
1.2	Mission Objectives	07/02/2014	Project Goals
1.3	Project Target	07/02/2014	Project Scope
1.4	Threat Model	07/02/2014	Project Scope
1.5	System Requirements	07/02/2014	Same as Desc.
1.6	User View and Requirements	07/02/2014	Same as Desc.
1.7	Transaction Requirements	07/02/2014	Same as Desc.
1.8	Case Studies (CS)	14/07/2014	Eval. of rival
1.8.1	CS: Facebook	14/07/2014	Eval. of rival
1.8.2	CS: GPG and E-Mail	14/02/2014	Eval. of rival
1.8.3	CS: Tor	14/02/2014	Eval. of rival
1.8.4	CS: 'aam' and mix networks	14/02/2014	Eval. of rival
1.10	Risk Assessment	14/02/2014	Same as Desc.
1.11	Anticipated Software	14/02/2014	Project Estimates
1.12	Anticipated Experiments and Evaluation	14/02/2014	Project Estimates
1.13	Anticipated Documentation	14/02/2014	Project Estimates
1.15	User View	14/02/2014	Same as Desc.
1.16	Gantt Chart	14/02/2014	Same as Desc.

Task ID	Task Description (Desc.)	Due Date	Deliverable
2	Project Design	14/03/2014	Design Segment
2.1	Research (Res.)	21/02/2014	Research Segment
2.1.1	Res: Database Languages	21/02/2014	Same as Desc.
2.1.2	Res: Programming Languages	21/02/2014	Same as Desc.
2.1.3	Res: Interfaces	21/02/2014	Same as Desc.
2.2	Designs (Des.)	07/03/2014	Design Segment
2.2.1	Des: Databases	28/02/2014	Same as Desc.
2.2.2	Des: Class Interfaces	28/02/2014	Same as Desc.
2.2.3	Des: Protocol	28/02/2014	Same as Desc.
2.2.4	Des: Architecture	28/02/2014	Same as Desc.
2.2.5	Des: Sequence Diagrams	28/02/2014	Same as Desc.
2.2.6	Des: Data Flow Diagrams	28/02/2014	Same as Desc.
2.2.7	Des: Class Diagrams	28/02/2014	Same as Desc.
2.2.8	Des: Server-side Interfaces	28/02/2014	Same as Desc.
2.2.9	Des: Client-side Interfaces	28/02/2014	Same as Desc.
2.2.10	Des: Server-side Protocols	28/02/2014	Same as Desc.
2.2.11	Des: Client-side Protocols	28/02/2014	Same as Desc.
2.2.12	Des: Server-side Pseudo-code	07/03/2014	Same as Desc.
2.2.13	Des: Client-side Pseudo-code	07/03/2014	Same as Desc.
2.3	Segment Review	10/03/2014	Design Segment
2.3.1	Evaluate Segment Quality	14/03/2014	N/A
2.3.2	Improve Segment	14/03/2014	Design Segment

Task ID	Task Description (Desc.)	Due Date	Deliverable
3	Implementation stage (Imp.)	28/04/2014	Imp. Segment
3.1.1	Imp: Architecture	21/03/2014	Work Environment
3.1.2	Imp: Architecture Docs	21/03/2014	Documentation
3.2.1	Imp: Target System (TS)	21/03/2014	Work Environment
3.2.2	Imp: TS Documentation	21/03/2014	Documentation
3.3.1	Imp: Databases	21/03/2014	Database
3.3.2	Imp: Database Documentation	21/03/2014	Documentation
3.4.1	Imp: Server-side Protocols	28/03/2014	Program function
3.4.2	Imp: Server Protocol Docs	28/03/2014	Documentation
3.5.1	Imp: Client-side Protocols	28/03/2014	Program function
3.5.2	Imp: Client Protocol Docs	28/03/2014	Documentation
3.6.1	Imp: Server-side Interface	04/04/2014	Interface
3.6.2	Imp: Server Interface Docs	04/04/2014	Documentation
3.7.1	Imp: Client-side Interface	04/04/2014	Interface
3.7.2	Imp: Client Interface Docs	04/04/2014	Documentation
3.8.1	Imp: Server-side Source Code	18/04/2014	Program
3.8.2	Imp: Client-side Source Code	18/04/2014	Program
3.9.1	Imp: Server Install Docs	18/04/2014	Documentation
3.9.2	Imp: Client Install Docs	18/04/2014	Documentation
3.10	Segment Review	28/04/2014	Imp. Segment
3.10.1	Evaluate Segment Quality	28/04/2014	N/A
3.10.2	Improve Segment	28/04/2014	Imp. Segment
Task ID	Task Description (Desc.)	Due Date	Deliverable
4	Project Portfolio	09/05/2014	Portfolio
4.1	Fabricate Reports	02/05/2014	Reports

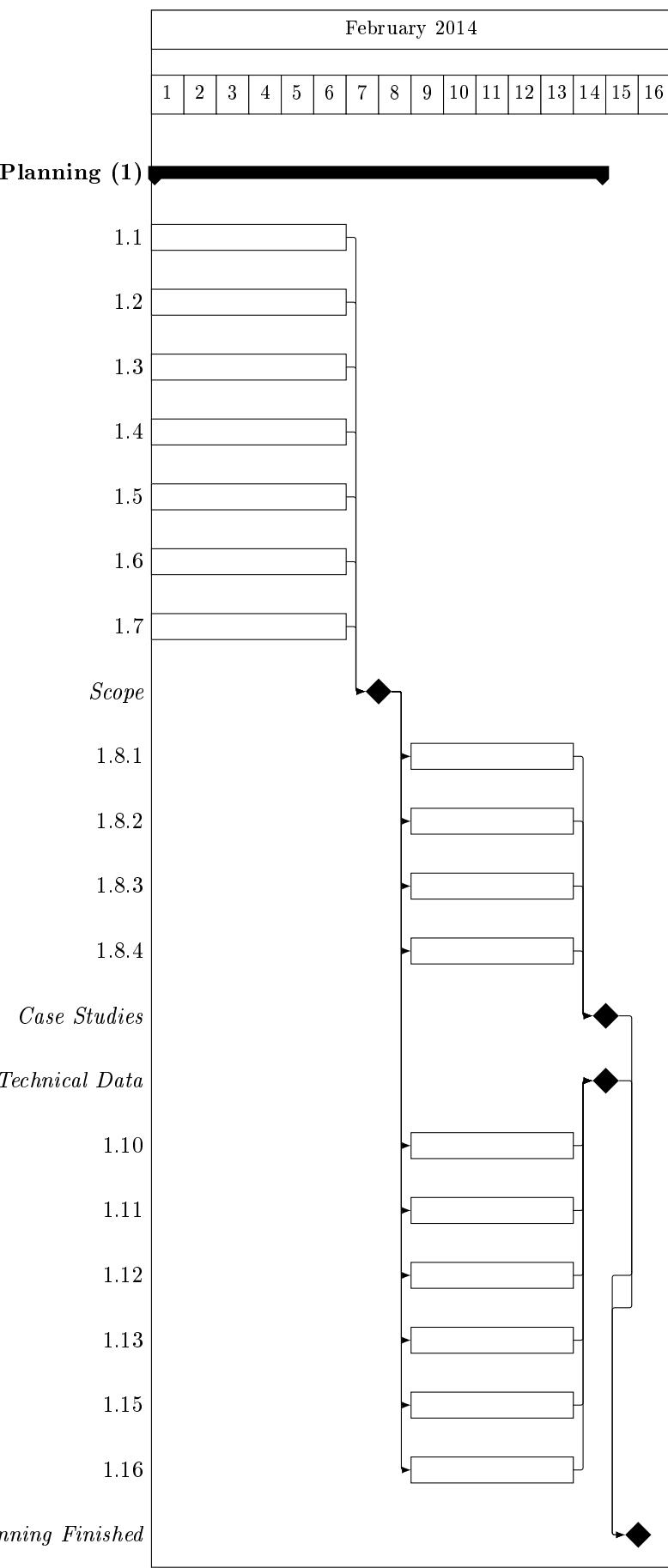
Chapter 18

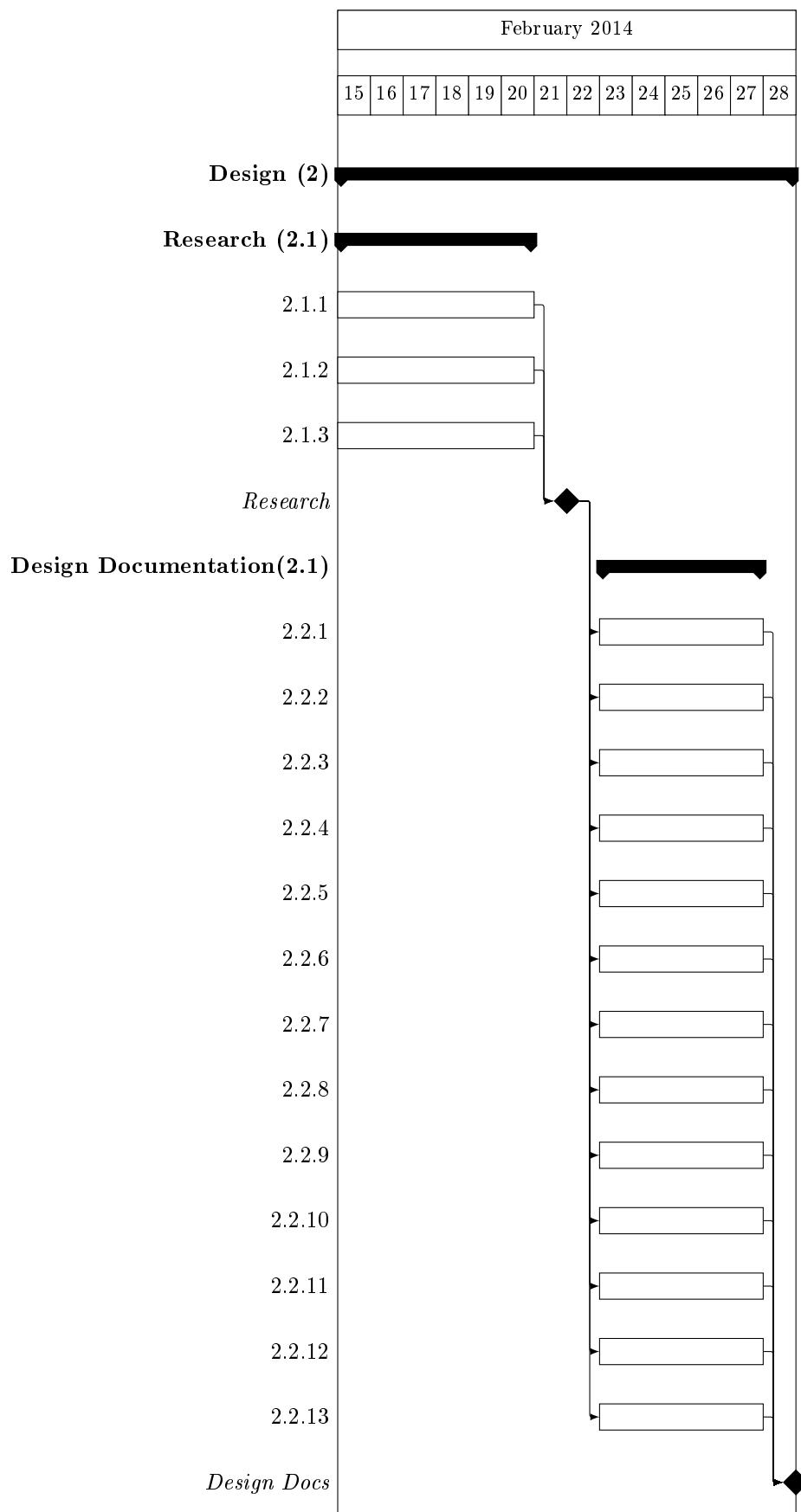
Gantt Chart

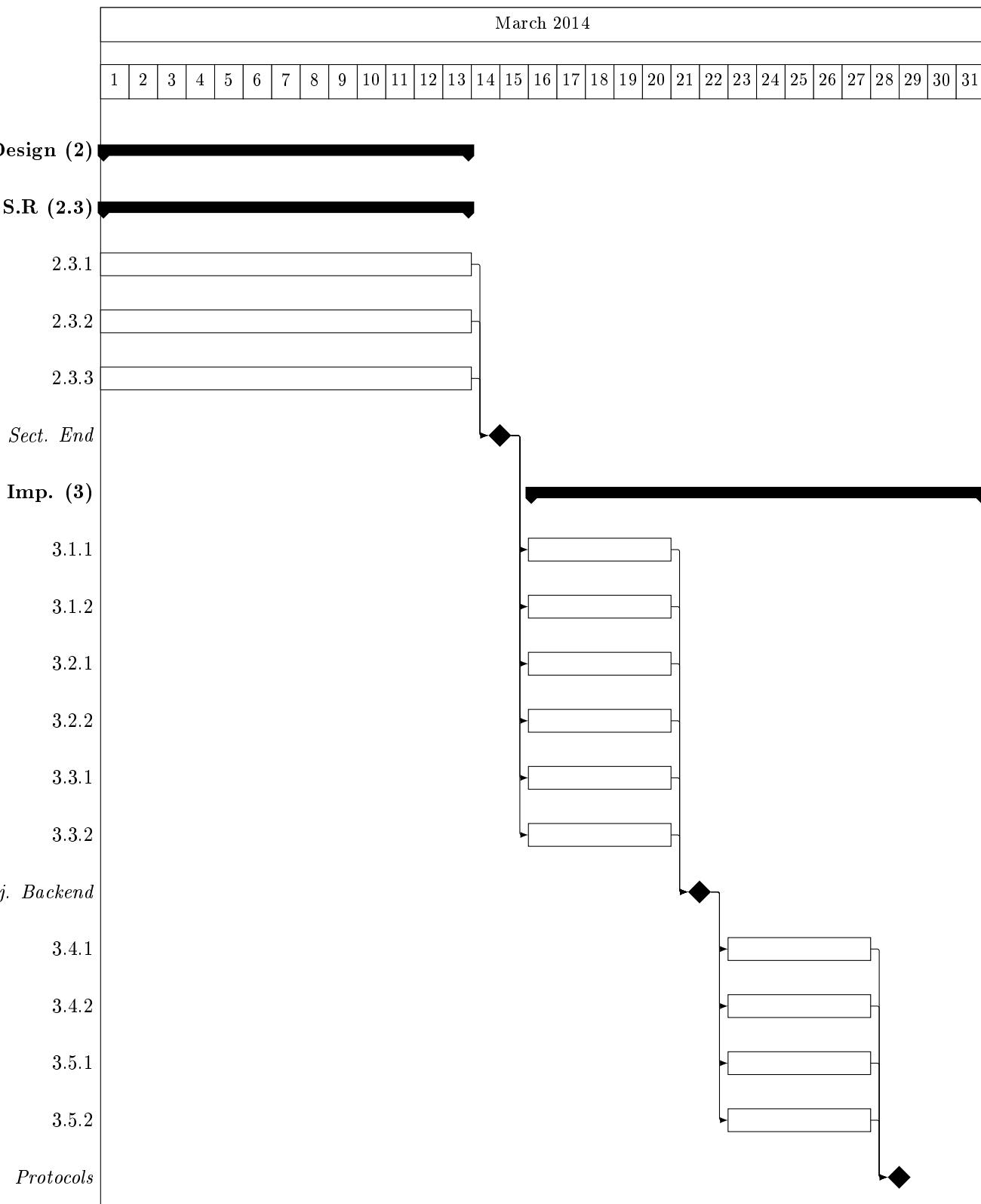
The following are Gantt Charts of the project. They are developed from looking at overall requirements for the project, and to act as a base for us to follow when developing the project.

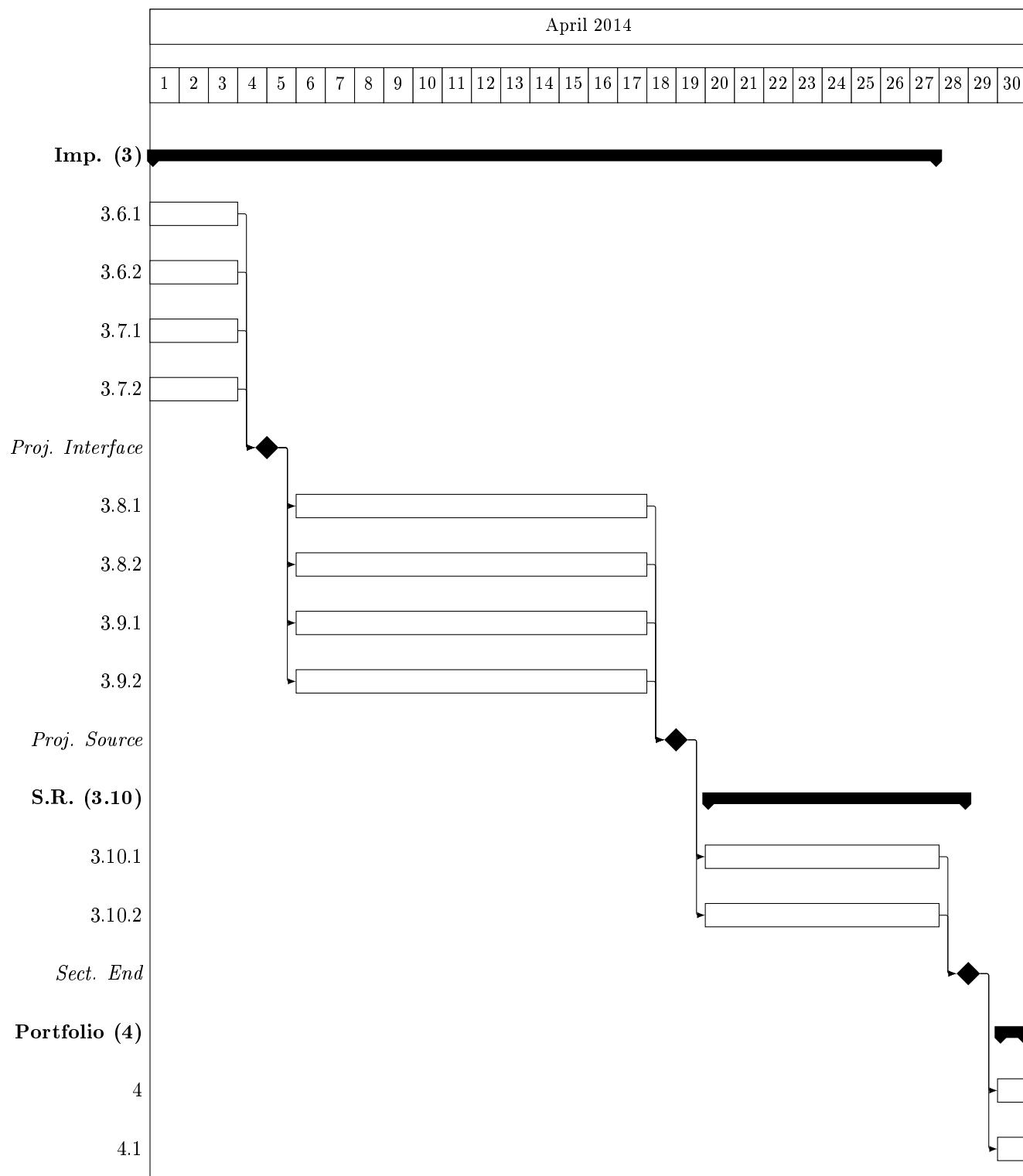
It is a graphical representation of the Task List documented prior to the charts. They have been split up either monthly or bi-monthly basis to allow acceptable formatting, due to differing workloads between months.

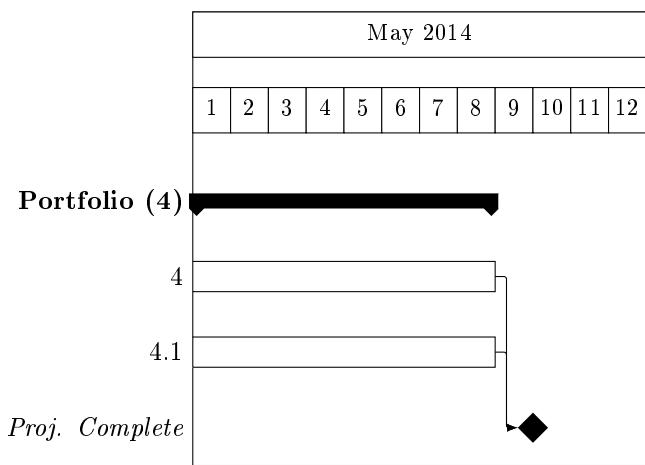
February is the most work intensive as it is providing the foundations of the project and is therefore detailed to reduce the risk of the project failing. April is the most lax of the working months (apart from May being project clean up) as the group members will be concentrating on the larger tasks as opposed to diverting attention between four or five tasks each, like in the Planning stage of the project.











Chapter 19

Risk Assessment

19.1 Parallel Tasks

A big concern for any project is the amount of tasks that will be performed simultaneously [3]. For every task that is carried out together, but potentially separate from each other, risk is increased - with more tasks making a more dramatic increase of potential failure for the project. For the planning section of this project we have performed a large amount of tasks simultaneously which may be detrimental to our quality of work later in the project.

In order to reduce or even eliminate the risk of too many parallel activities, the project should be planned using a Gantt chart to reduce the amount of tasks being performed simultaneously, and have more milestones within the project. This will help effectively split up the project into more manageable sections, which will not only make the project seem simpler to complete, but will improve the monitoring capabilities of the project as well.

19.2 Group Work

Working within a group can make deliverable dates difficult to achieve. This can be due to a lack of communication, unavailability of party members or an incapability to meet deadlines for some of the members. A meeting of minds also includes an assemblage of work ethics. Because of this, work may grind to a halt as members argue over personal yet trivial matters such as formatting documents or a varying opinion on what is classed as 'enough work' for a task.

Combating the disadvantages of working as a group can be difficult. As some problems are part of a group unable to function either properly or efficiently together, this can be the breaking factor of the project. This is a risk that cannot be eliminated but can be reduced. A way of minimising the amount of damage that the risk will do would be to have a centralised form of contacting members of

the group - examples being a website or using a revision control system such as 'Apache Subversion' or 'Git', will give a common area for the group to look for potential absences or reasons for reduce work output from members.

The best way to reduce the risk of differing qualities of work between the group would be to define a standard of work between the group - such as the layout of source files in programming languages, or a house style for formal documents as part of the group's external identity. Having this be available to the group in some form, such as in a text file within a shared area will allow the group to refresh their memories of parts of the set standard that they wouldn't follow otherwise.

19.3 Deadlines

Deadlines are the final day or dates that an object needs to be completed by. Sometimes within a project the deadline may be overstepped due to any of the risks mentioned within this document, which can lead to something small such as being berated by the project leader or something serious such as a breach in contract with the client. For these reasons, deadlines need to be adhered to so that the project can continue on schedule.

Reducing the risk of deadlines are important, especially for those that are not capable of monitoring their time effectively. By providing deadlines as a range of dates as opposed to a singular date, there is increased flexibility within the project and it gives some people more time to finish their work if it is required. By using a range of dates the group can finish on the beginning of the deadline range - a sort of pseudo-deadline - meet up and discuss whether alterations need to be made on the work, and then use the remainder of the time until near the end of the deadline range to perform them.

19.4 Scope

The scope is what the project will be encompassing and therefore is one of the most important sections as it defines what you'll be doing for the entirety of the project. That's not the only risk associated with the scope [21]. There is also scope creep, which is when the scope grows to cover more work than the project originally intended, often without an increase in resources matching the higher load on the project deliverables. Performing estimates on the scope, as well as anything else in the project, can be inaccurate as you are essentially guessing the near future, which is difficult at the best of times.

To minimise the risk placed upon the scope, it is best to define what exactly is required of the project before any work takes place on the deliverables. For example it is best to define an encryption method for a project at the beginning and sticking to it, rather than changing the method which may require a different implementation, creating more work. If at a later phase

ambiguities appear in the scope, a meeting to define or even redefine these points should occur before any more work is carried out on the offending article, reducing the amount of change to the project that shall occur.

19.5 Change Management

Change Management is the application of a structured process and set of tools for leading the people side of change to achieve a desired outcome [14]. Problems that are associated with Change Management include conflicts which occur between stakeholders, as they may be disagreeing in how the project should move forward. Assuming that an irreparable state has befall the project due to a drastic amount of changes that have been placed upon the project, or even ambiguous or inaccurate changes being added onto the project [21]. All of these can amount into an increase in workload, or a decrease if the targets haven't been properly defined.

To reduce the amount of risk involved with Change Management, communication and clear definition on what the project needs to perform is required. Stakeholders should be as detailed as possible at every stage so that no ambiguity is caused, or cleared up if any does occur.

19.6 Stakeholders

Stakeholders are people that have an interest in the project, whether they are the members of the group, the group's monitor/superior or the target audience of the project. Some of the problems that Stakeholders cause for the project members include:

- Losing interest: if they become uninterested with the project then they may back out, which can be dangerous for the project if they were providing any form of input, such as experience in the target field or economic support.
- Stakeholders becoming disillusioned: They are unaware of what the deliverables will be or have a twisted view on what and how the final product will perform its intended purpose.
- Quality Risk: Stakeholders may give ambiguous input both accidentally or on purpose, depending whether the Stakeholder wants the project to fail or not [21].

The best way to reduce the amount of risk involved with Stakeholders would be to keep them informed of the project's current status through external communication such as e-mail, and through meetings so that the team can personally inform the Stakeholder with relevant information, which should ease their mind of any apprehensive thoughts about the project [6].

19.7 Platforms

The main risk in Platforms would be the difference between the chosen development platform and the target market's system. The change between executable files for different operating systems are usually great enough so that a separate executable is required for each distinct operating system. What may also cause problems, especially with low-level programming, would be differing architectures, hardware sets and how the system reads commands [23]. Another problem with platforms would be whether the required software for the project is installed, such as any required run-time environments or files which are needed to use Structured Query Language databases.

This risk can be eliminated if platform-independent code is used - such as the Java Programming Language [13]. This would mean that no changes in implementation would be needed, and database functionality could occur within the platform-independent environment if need be. Otherwise to reduce the amount of risk involved with the varying systems that the target audience may own, compiling the source on different virtual systems to create executables for the many various platforms available would suffice. Of course, this can be mitigated by choosing to not support other systems in favour of only allowing the development platform and its Operating System to be supported.

19.8 Integration

The integration of the project can be high risk due to a couple of factors:

- The intended environment is incompatible or unavailable
- Incomplete testing means the final product may be buggy
- Final product doesn't work (e.g. bad link to database)
- Product lowers efficiency due to learning curve [21]

In order to combat the risks involved in implementation, having a set testing day in an isolated environment can allow the completed builds of the project to be evaluated before being given to the target audience. This will allow the checking of compatibility with the system as well as in-house bug testing. A manual or help section could be implemented into the system so that the learning curve is not as steep compared to not having such resources.

19.9 Requirements

Requirements are not just a list of functional needs and wants but also the constraints on the project as well. However, there are similar risks involved in the requirements, such as generalisation,

ambiguity or even being incomplete. Another risk to do with requirements is whether they align with the design factor or not.

An example would be having both 'fast processing' and 'system independence' as requirements; C++ is faster but Java is independent of platform and although speed may not be an issue with smaller data, larger chunks of data will undoubtedly have an effect on interpreted code [17].

To minimise the risk with requirements, communication between group members and stakeholders is needed; making sure that the requirements and the scope are in line with each other, and that any suggested changes are properly handled with little to no ambiguity. Choosing a design structure and sticking to it is also beneficial to the project. Reducing the workload of the implementation can help towards minimising the risks of requirements and the program, such as removing old data that is no longer needed upon the program's start-up.

19.10 Authority

Without distinct authority within the project, risks can become apparent. If the members of the project do not have the correct privileges on the target system to perform what is required, work output slows or even stops until the matter is resolved. Another risk would be misguided authority; where the team is unclear who has been given the authority to perform a task and therefore there are multiple members allocating the same task to themselves, which will slow down the efficiency of the team due to duplicated work.

Lowering the negative impact of Authority is done through the use of clear definitions. Allocating work to project members and centralising a form of 'to-do' list so that project members can look up what has been assigned to them. Another way of reducing the amount of inefficiency caused by problems with authority would be to make sure the permissions are correctly set up on both the testing and target systems.

19.11 External

There are a couple of external factors which may impact the project in a negative manner. The first being any legal restrictions. This is important as there is a chance that the final product may be used in a location that differs to the geographical area that it was developed in. For example there is a law within the UK which requires that you must provide encryption keys under certain circumstances to the UK authorities [4][22]. In the USA however, it is something of a grey-area, as giving up encryption keys could violate the fifth amendment, as doing so could give incriminating evidence against yourself:

⁴unlike surrendering a key, disclosing a password reveals the contents of oneâŽs mind

and is therefore testimonial.' [27]

Not only is the law a big risk in projects, but also nature. If you are situated where natural disasters can happen or otherwise things such as heavy weather occur, this can reduce the work flow by denying the team members access to their workspace. Another factor that is external is the changing of technology. Updates to programming languages can lead to deprecated functions or newer operating systems may not be capable of running the same software as their previous iterations, meaning an increased amount of work to keep the software compatible with the target system.

Reducing the amount of risk caused by external factors is difficult as the project team have little to no influence upon them. For example the team cannot bypass any laws that govern the area that the program will be used in, so they must be adhered to as part of the constraints of the project. Natural disaster cannot be stopped, but if you are able to, bringing some of the work back so you could work on it during bad weather may reduce the impact that said weather will have on the project. To reduce the damage caused by software depreciation it is ideal if the functionality coded in the project is not old, or otherwise buggy, so that maintaining or updating the software will require less work.

19.12 Project Management

Project Management, or rather a lack of, can also be a risk to the endeavours of the team. If the group has been asked to reduce or combine the amount of stages in the System Development Life Cycle (SDLC), this can increase the risk of the project failing because it leaves more room for error; combining the stages will often cause a decrease in quality, as less resources are being dedicated to a particular section of the project. A lack of Project Management will also be seen as a high risk because of how difficult it is to monitor a project and its success without these tools.

To reduce the risk that Project Management will apply upon the project, a formal methodology, such as the 'waterfall' method could be implemented. This would however reduce inefficiency as the output needs to be moderated and cleared before the start of the next stage in the SDLC can occur. On the other hand an informal methodology would increase the risks, but may potentially allow the project to be completed within a smaller time frame and to the same standard.

19.13 User Acceptance

Just because a project has been made for a target audience doesn't mean that *that* audience will like it. During testing the target market may reject the initial builds of the project due to the way

it does or does not work, or the look of the project could mean that it is unwieldy to use, whether it is due to low quality or the interface being anti-intuitive.

The main method of reducing the risk pre-emptively is to perform research on any currently available software that achieve similar goals to the project's. By doing this you can find out what users are acquainted with and create a similar yet unique design, or use the competitors as a way of highlighting what is wrong with the current market and create something entirely different. Another method which does require more work is to take in user feedback during testing and implement their suggestions for the look of the project, or the inner mechanics if they have the knowledge to suggest improvements.

19.14 Conclusion

In order to reduce the risk of the project as a generalisation, it is suggested that you:

- Have a centralised communication system used by all members - this reduces all communicative related risks.
- Define team objectives and allocation clearly - this reduces the authority-based risks as well as any that are communicative.
- Define a target system for development - other types of platform can be supported at a later date should the need arise.
- Create and uphold a work ethic to be followed by everyone - this helps to maintain a standard of quality throughout the project.
- Testing should be first on each individual module/deliverable, then as a whole. This improves bug catching and helps monitor the quality of the project.
- Choose a methodology and follow it - this creates a standard of work ethics which will give a layout as well as structure to the project.

By following these pointers a moderate amount of risk can be mitigated with little need for concern. Do note that the legality of the project in differing countries should be researched and followed, should the project be in use within that country.

Part II

Design

Chapter 20

Proposal Summary

The project, a security based social media network, will have multiple components to be investigated and used in this design section. The key critical components to be looked at consist of:

- Database
- Client
- Client GUI
- Server
- Server GUI
- Mobile GUI (future work)

Of each of these components we should look at how they will impact their respective uses in order to best make use of their full functionality. We will look at multiple possible and practical solutions for the above criteria, making sure the best solution is chosen. We will also look at possible work in the future, or any areas to continue with into the coming stages.

The requirements section has helped so far through analysis of existing social media networks and how they have implemented their networks, along with how their interfaces react to the user.

Chapter 21

Architecture

21.1 Network Architecture

Turtlenet is a centralized service, whereby a large number of clients connect to a single server which provides storage, and facilitates communication between clients.

Due to the inherently limited network size (5-50K users per server depending on percentage of active participants vs consumers and local internet speeds) we recommend that servers serve a particular interest group or geographic locality.

Clients send messages to, and only to, these central servers. Due to the fact that all messages (except CLAIM messages, see client-server/client-client protocols for details) are encrypted the server does not maintain a database, it cannot; rather clients each maintain their own local database, populated with such information to which they have been granted access.

When a client wishes to send a message to a person, they encrypt the message with the public key of the recipient¹ and upload it to the server. It is important to note that all network connections are performed via Tor.

When a client wishes to view messages sent to them, they download all messages posted to the server since they last downloaded all messages from it, and attempt to decrypt them all with their private key; those messages the client successfully decrypts (message decryption/integrity is verified via SHA256 hash) were intended for it and parsed. During the parsing of a message the sender is determined by seeing which known public key can verify the RSA signature.

Due to the nature of data storage in client-local databases, all events and data within the system must be represented within these plaintext messages. This is achieved by having multiple types of messages (see client-client protocol).

¹using RSA/AES, see protocol for details

21.2 System Architecture

The system has a number of modules which interact with one another via strictly defined interfaces. Each module has one function, and interacts as little as possible with the rest of the system. The modules and their interactions are shown below. NB: $a \rightarrow b$ denotes that data passes from module a to module b, and $a \leftarrow b$ similarly denotes that data passes both from a to b and from b to a.

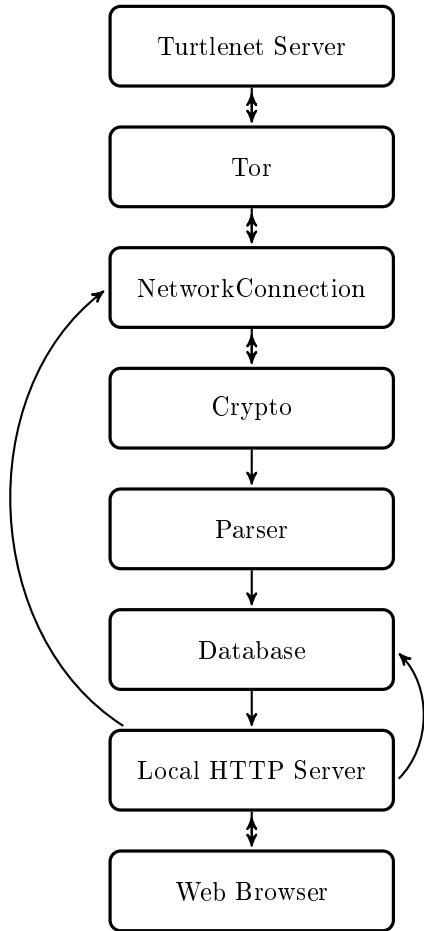


Figure 21.1: Module Interaction

21.3 Data Flow Diagram

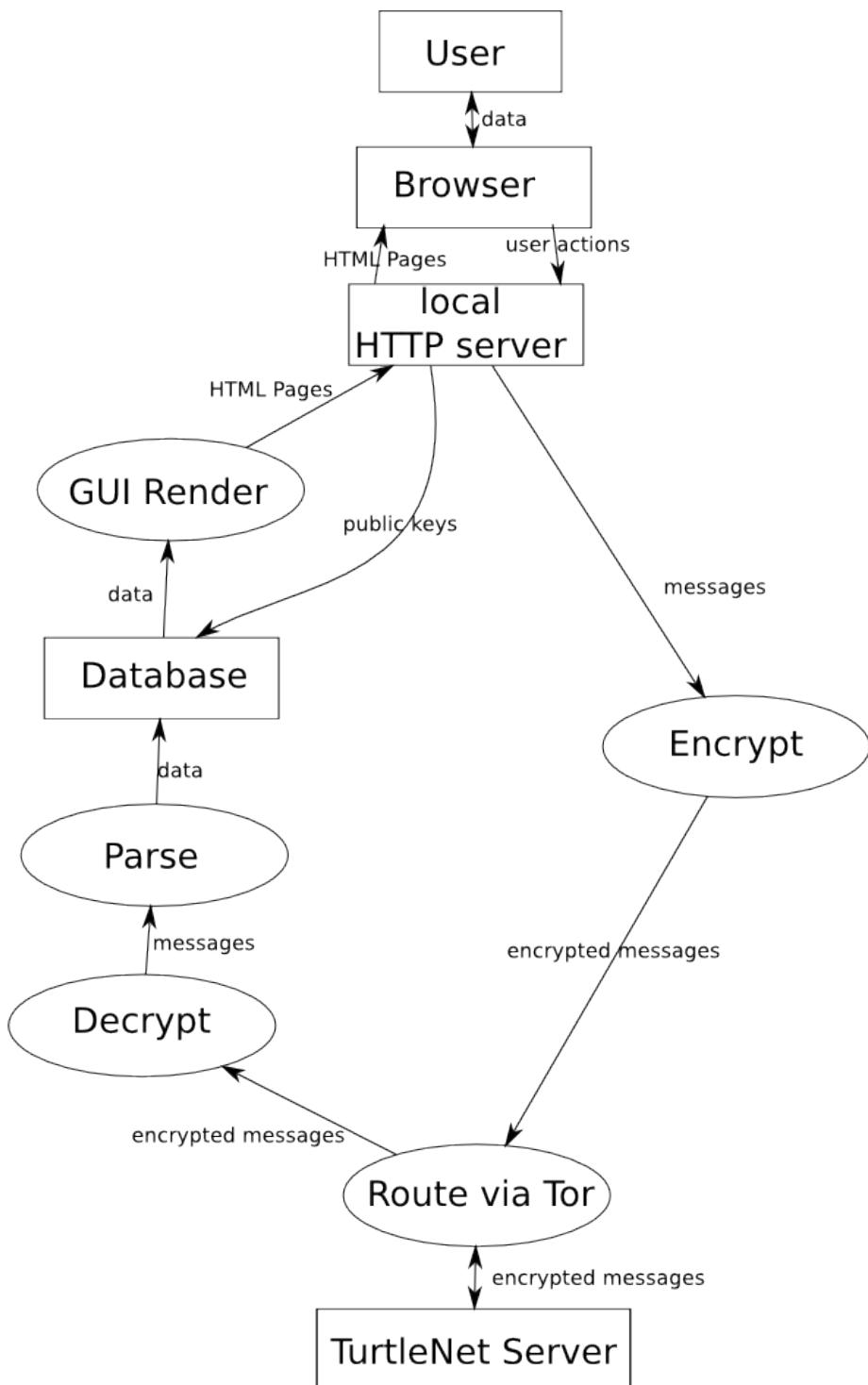
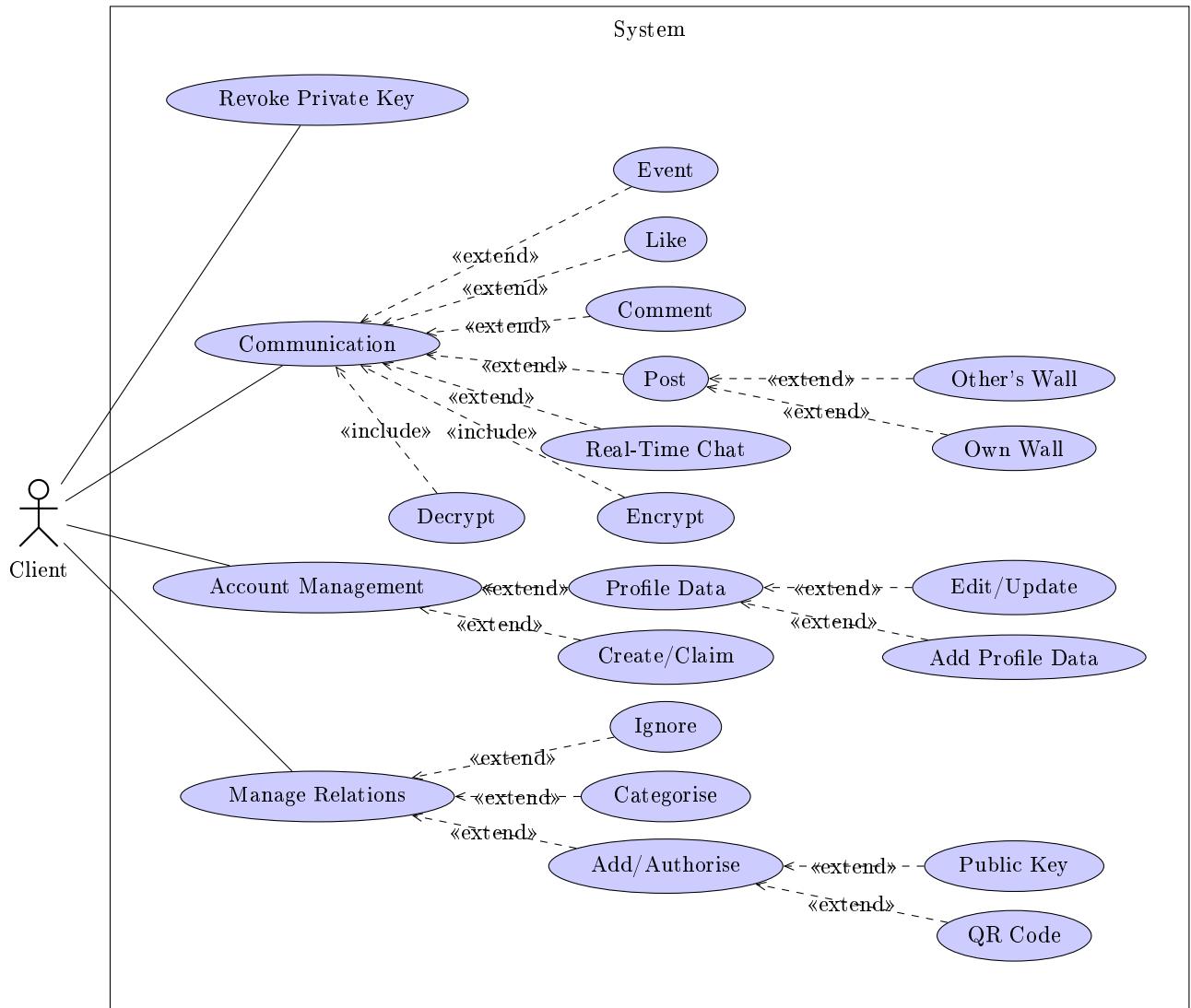


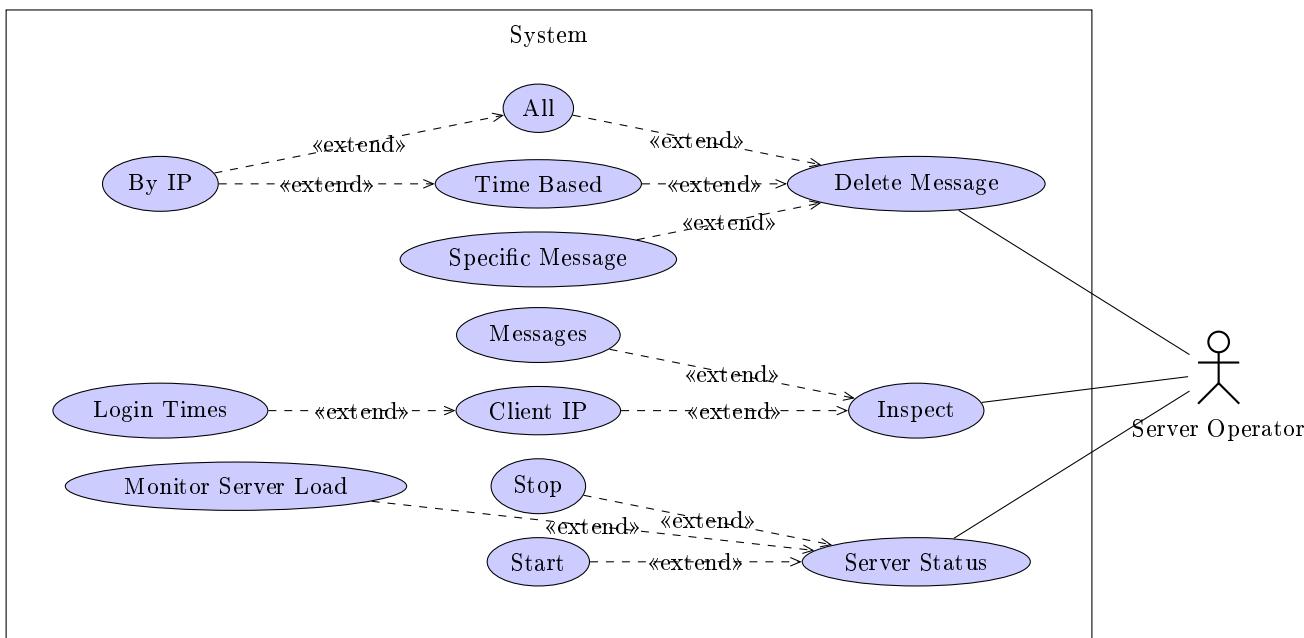
Figure 21.2: Data Flow Diagram

Chapter 22

Use Case Diagrams

Here we have a use case diagram displaying an actors interaction with our system. It shows the functionality available to both the client, and the operator. We also have a sequence diagram to augment the use case diagrams.





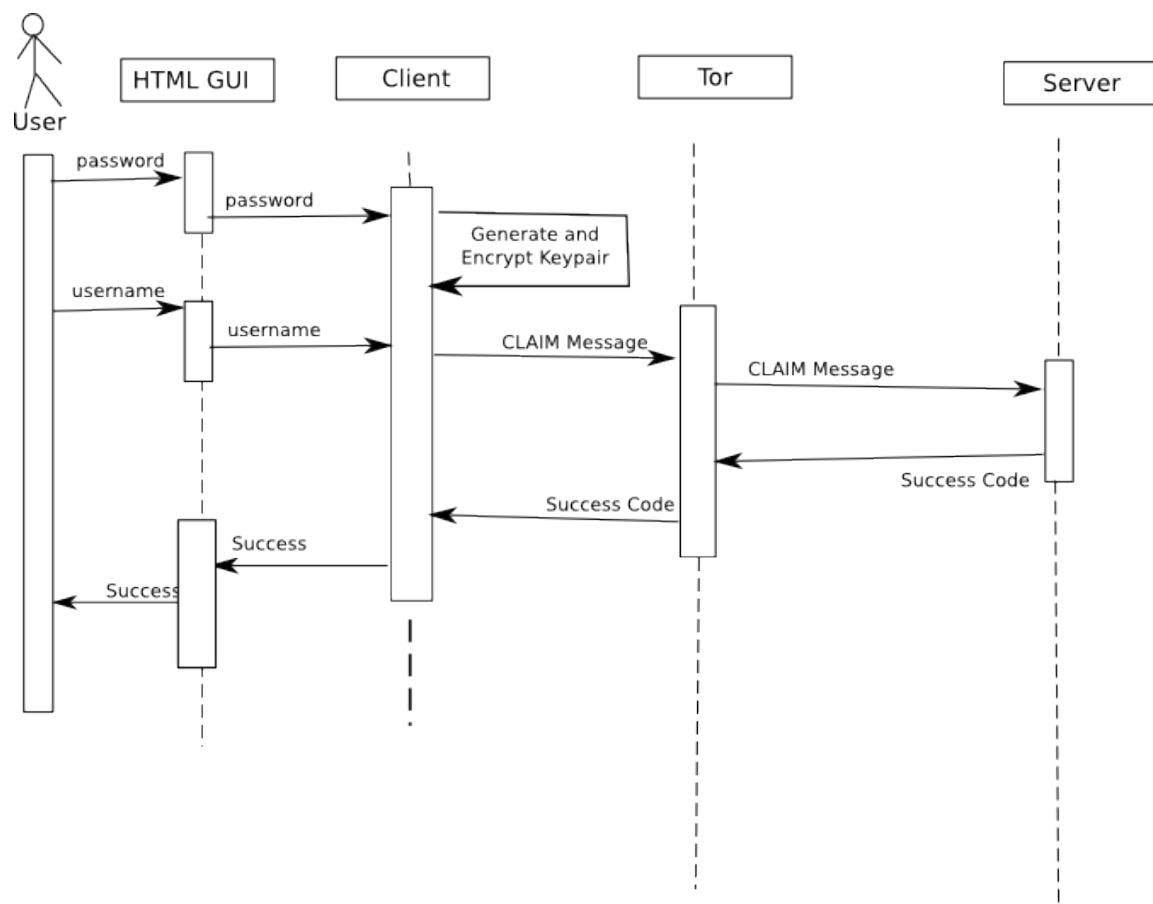


Figure 22.1: Sequence Diagram - Registering

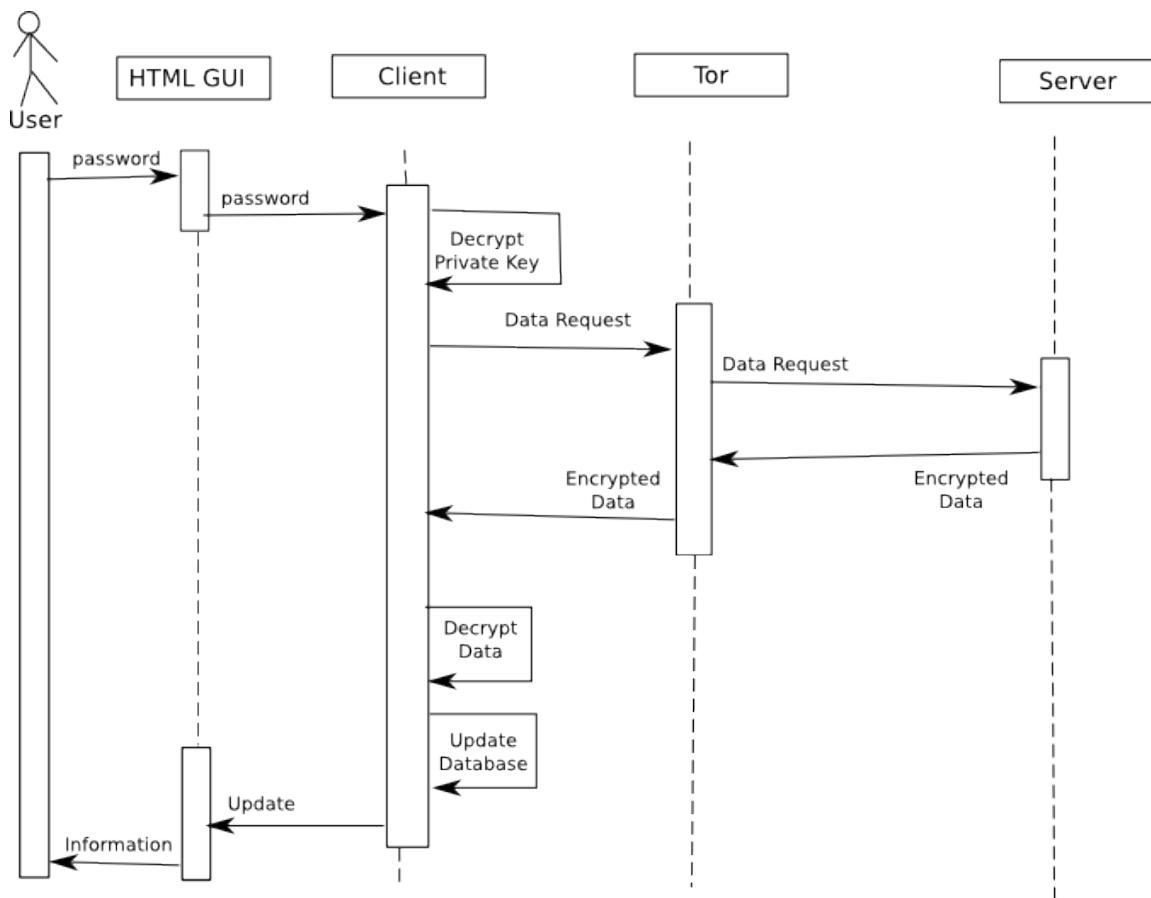


Figure 22.2: Sequence Diagram - Recieving Data

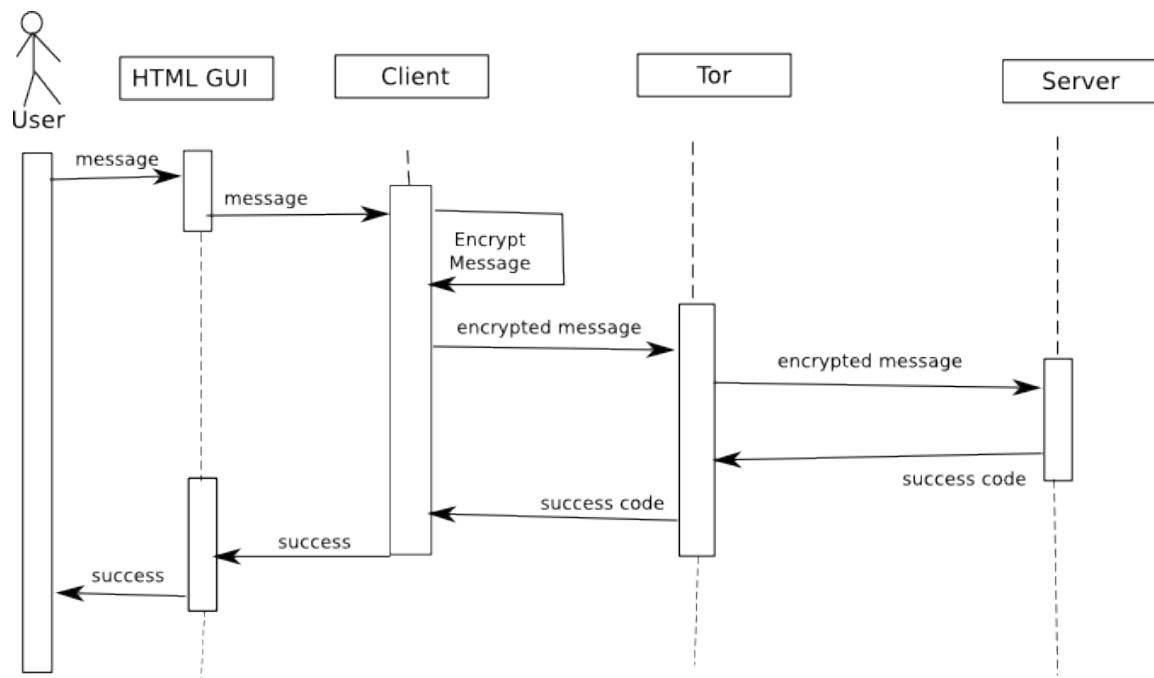


Figure 22.3: Sequence Diagram - Sending Data

Chapter 23

Protocol

23.1 High Level Summary of Protocol

Creating an account is done by generating an RSA keypair, and choosing a name. An unencrypted (but signed) message is then posted to the server associating that keypair with that name. In this way, by knowing the public key of someone, you may discover their name in the service, but not vice versa.

Connecting for the first time Every unencrypted message stored on the server is downloaded(signed nicknames and nothing more). At this time the local database contains only signed messages claiming usernames. The public keys are not provided, these are of use only when you learn the public key behind a name. The rationale for not providing public keys is provided in the section regarding adding a friend. Messages posted after your name was claimed will require downloading too, as once you claim a name people may send you messages. It's worth noting that messages from before you connected for the first time are now downloaded because they can not have been sent to you (with a compliment client) if someone retroactively grants you permission to view something they publish it as a new message with an old timestamp; the sole exception to this is when you connect using a new device, in which case all messages since you first claimed a name will be downloaded.

Connecting subsequently The client requests every message stored on the server since the last time they connected up to the present. Decryptable messages are used to update the local DB, others are discarded.

Continued connection During a session the client requests updates from the server every

0.5-5 seconds (configurable by the user).

Adding a friend is performed by having a friend email (or otherwise transfer) you their public key. This is input to the client, and it finds their username (via public posting that occurred when registering). You may now interact with that person. They may not interact with you until they receive your public key. Public key transferral will be performed via exchanging plaintext base64 encoded strings, or QR codes. The user will be prompted, after retrieving the username of the user, to categorise them.

Talking with a friend or posting on your wall is achieved by writing a message, signing it with your private key, and encrypting one copy of it with each of the recipients public keys before posting it to the server. The client prevents one from posting a message to someone's public key if they have not claimed a nickname.

Posting to a friends wall, commenting and liking may be requested by sending a EPOST/CMNT/LIKE message to the friend (upon whose wall/post you are posting, commenting or liking), when that friend logs in they will receive your request, and may confirm or deny it. If they confirm then they take your (signed) message and transmit it to each of their friends as previously described. Given that authentication is entirely based on crypto signatures it doesn't matter that your friend relays the message. This is required because it is impossible for one to know who is able to see the persons wall, post, or comment upon which you seek to post, like, or comment.

23.2 Client-Server Protocol

The client-server architecture is necessarily simple.

The client connects to the server, sends a single command, receives the servers response and then disconnects. The following shows commands sent by the client, and the servers action in response.

command	purpose	servers action
t	get the server time	sends back the current time (unix time in milliseconds)
s <i>utf-8_text</i>	send messages	the text sent is stored on the server
get <i>ms_unix_time</i>	get new messages	every message stored since the given time is sent
c <i>utf-8_text</i>	claim a username	the text sent is stored on the server, with a special filename

Table 23.1: Client-Server Protocol

Every command is terminated with a linefeed. Every response from the server will be terminated with a linefeed. The last line sent by the server will always be "s" for success, or "e" for failure (this is omitted from the above table).

CLAIM messages (sent with c) will be parsed by the Message class and the username extracted for use in a filename. The filename of claim messages is as follows $<\text{unix_time_in_ms}>_<\text{username}>$; the filename of all other messages is as follows $<\text{unix_time_in_ms}>_<\text{SHA256_hash}>$.

23.3 Client-Client Protocol

23.4 Summary

All client-client communication is mediated by the server. When one client wishes to send a message to another it encrypts the message with the public key associated with the recipient and uploads it to the server. When one client wishes to receive a message it downloads all new messages from the server and parses those it can decrypt. This is performed in order to hide who receives a message. All messages except CLAIM messages are encrypted. Multiple recipients imply multiple messages being uploaded, this is taken for granted in the text which follows.

23.5 Message Formatting

23.5.1 Unencrypted Messages

Messages have a command (or type), which specifies the nature of the message; messages have content, which specifies the details of the message; messages have an RSA signature, which authenticates the message; messages have a timestamp, which dates the message down to the millisecond, the time format is unix time in milliseconds.

Messages are represented external to the system as utf-8 strings, and internally via the Message class. The string representation is as follows:

$$<\text{command}>|<\text{signature}>|<\text{content}>|<\text{timestamp}>$$

Backslashes are literal, angle brackets denote placeholder values where data specific to a message is placed.

An example follows:

```
POST\<signature>\Hello, World!\1393407435547
```

backslashes in message content are escaped with another backslash, signatures are base64 encoded SHA256/RSA signatures of the content of the message concatenated with a decimal string representation of the timestamp. All text is encoded in UTF-8.

23.5.2 Encrypted Messages

Encrypted messages contain the AES IV's; the RSA encrypted AES key; and the AES encrypted message.

Messages are encrypted by encoding the entire message to be sent with UTF-8; encrypting the message with a randomly generated AES key; encrypting the AES key with RSA; encoding the RSA encrypted AES key in base64; encoding the (random) AES initialization vectors in base64 and concatenating these three parts with a backslash between each. The format follows:

```
<AES IV>\<RSA encrypted random AES key>\<AES encrypted message>
```

Backslashes are literal, angle brackets denote placeholder values where data specific to a message is placed.

23.6 Claiming a Username

Each user (keypair) should claim one username. Uniqueness is enforced by the server, and so not relied upon at all. Usernames are useful because public keys are not human readable. In order to claim a username, one must send an unencrypted CLAIM message to the server. The format follows:

```
CLAIM\<signature>\<username>\<timestamp>
```

23.7 Revoking a Key

If a users private key should be leaked, then they must be able to revoke that key. This is done by sending a REVOKE message to the server. All content signed by the private key after the stated time will be flagged as untrusted. The format follows:

```
REVOKE\<signature>\<time>\<timestamp>
```

23.8 Profile Data

Users may wish to share personal details with certain people, they may share this information via profile data. Profile data is shared using PDATA messages. A PDATA message contains a list of fields, followed by a colon, followed by the value, followed by a semicolon. The format follows:

```
PDATA\<signature>\<values>\<timestamp>
```

The format for values follows:

$<field>: <value>; \dots$

An example follows:

PDATA\<*signature*>\name:Luke Thomas;dob:1994;\<*timestamp*>

23.9 Inter-User Realtime Chat

Users can chat in real time, this is achieved by sending a CHAT message to all people you wish to include in the conversation. This message includes a full list of colon delimited public keys involved in the chat. The format follows:

CHAT\<*signature*>\<*keys*>\<*timestamp*>

The format for keys follows:

$<key>: <another_key>; \dots$

An example follows:

CHAT\<*signature*>\<*key1*>:<*key2*>\<*timestamp*>

Following the establishment of a conversation, messages may be added to it with PCHAT messages, the format follows:

PCHAT\<*signature*>\<*conversation*>:<*message*>\<*timestamp*>

Whereby $<conversation>$ denotes the signature present on the establishing message. An example follows:

PCHAT\<*signature*>\9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08:First!\<*timestamp*>

23.10 Posting to own wall

When a user posts to their own wall they upload a POST message to the server of the following format.

POST\<*signature*>\<*message*>\<*timestamp*>

The format of message is merely UTF-8 text, with backslashes escaped with backslashes.

An example follows which contains the text "Hello, World!", a newline, "foo \\bar\\baz":

POST\<*signature*>\Hello, World!
foo\\bar\\\\baz\<*timestamp*>

23.11 Posting on another users wall

A user may request to post on a friends wall by sending them an FPOST message, the poster may not decide who is able to view the message. The format is identical to that of a POST message, except for the command and singular recipient. An example follows:

```
FPOST\<signature>\Hello, World!\<timestamp>
```

Upon receipt of an FPOST message the friend is prompted by the client to choose whether or not to display it, and if so who may view it. Once this is done the friend reposts the message with the command changed to POST instead of FPOST as they would post anything to their own wall. This works because authentication is entirely based on RSA signatures so in copying the original signature the friend may post as the original author provided they don't alter the message (and thus its hash and required signature).

23.12 Commenting

Commenting works similarly to posting on another's wall, so an explanation of details of how it occurs is not provided (see prior section). The only difference is the format of a CMNT message from an FPOST message. The format of a CMNT message is as follows:

```
CMNT\<signature>\<hash>: <comment>\<timestamp>
```

Where *<hash>* denotes the hash of the post or comment being commented upon. An example comment follows:

```
CMNT\<signature>\v/sXfb3DG2qT2k2hXIH4csJy1yEG+TANRbbxQw1VkBSE=: Yeah, well,  
that's just like, your opinion, man.\<timestamp>
```

23.13 Liking

Like messages are identical to comments except for the command and the fact that no ":"*<comment>*" follows the hash. An example like follows:

```
LIKE\<signature>\v/sXfb3DG2qT2k2hXIH4csJy1yEG+TANRbbxQw1VkBSE=\<timestamp>
```

23.14 Events

A user may have the client remind him of an event by alerting him when it occurs. A user may inform others of events, and they may choose to be reminded about them. When a user creates

an event just for themselves they just create a normal event and only inform themselves of it. An event is created by posting an EVNT message to the server. The format follows:

EVNT\<*signature*>\<*event_start_time*>: <*event_end_time*>: <*event_name*>\<*timestamp*>

An example follows of a reminder for bobs birthday which occurs on the 14th of January, the event was created on the second of January:

EVNT\<*signature*>\1389657600000: 1389744000000: bobs birthday\1388676821000

Chapter 24

Class Interfaces

24.1 Class Interfaces

The following is a description of the public functions of all public classes. Many classes have inner private classes they use for convenience, however to simplify interaction between parts of our system ('modules') we have very few convenience classes.

return	function description
void	main() (static) starts the server

Table 24.1: Server

return	function description
void	main() (static) constructs and starts all necessary classes and threads, runs the main loop

Table 24.2: Client

return	function	description
N/A	NetworkConnection()	Constructs a NetworkConnection and connects to the given URL (through tor)
void	run()	periodically download new messages until asked to close, downloaded messages are stored in a FIFO buffer
void	close()	kills the thread started by run()
boolean	hasMessage()	return true if there is a message in the buffer, false otherwise
String	getMessage()	return the oldest message in the buffer
boolean	claimName()	claim a given username, returns true on success, false otherwise
void	revokeKeypair()	revokes your keypair
void	pdata()	adds or updates profile information
void	chat()	begins or continues a conversation
void	post()	post a message to your wall
void	fpost()	post a message to a friends wall
void	comment()	comment on a comment or post
void	like()	like a comment or post
void	event()	create an event

Table 24.3: NetworkConnection

return	function	description
boolean	keysExist()	(static) return true if the user has a keypair, false otherwise
void	keyGen()	(static) generate a keypair for the user
PublicKey	getPublicKey()	(static) returns the users public key
PrivateKey	getPrivateKey()	(static) returns the users private key
String	sign()	(static) returns an RSA signature of the passed string
boolean	verifySig()	(static) returns true if author signed msg, false otherwise
String	encrypt()	(static) returns an encrypted message constructed from the passed parameters
Message	decrypt()	(static) decrypts the passed string, returns the appropriate message, on failure a NULL message is returned
String	base64Encode()	(static) base64 encodes the passed data, returns the string
byte[]	base64Decode()	(static) base64 decodes the passed data, returns the byte[]
String	encodeKey()	(static) encodes a public key as a string, returns that string (X509)
PublicKey	decodeKey()	(static) decodes a public key encoded as a string, returns that public key(X509)
String	hash ()	(static) returns the SHA256 hash the the passed string as a hex string
int	rand ()	(static) returns a pseudorandom value <= max and >= min

Table 24.4: Crypto

return	function	description
void	parse()	(static) parses a sting message, records parsed data in the database

Table 24.5: Parser

return	function	description
void	addClaim()	adds a username CLAIM message
pair<string,string>[]	getClaims()	gets all CLAIMs to usernames
string[]	getUsernames()	gets all usernames
void	addRevocation()	adds a keypair revocation
pair<PublicKey, long>[]	getRevocations()	gets all revocations
boolean	isRevoked()	returns the time a key was revoked, if the given key has not been revoked then 0 is returned.
void	addPData()	adds (or amends existing) profile data
string	getPData()	gets the specified piece of profile data for a specified user
void	createChat()	creates new chat
pair<string,string>[]	getChat()	returns messages from a given chat
void	addToChat()	adds a post to a given chat
void	addPost()	creates new post, on your or another's wall
pair<string,string>[]	getPosts()	gets all posts either within timeframe, or from certain people within a timeframe
void	addComment()	adds a comment onto post or comment
pair<string,string>[]	getComments()	gets all comments for a post or comment
void	addLike()	likes a post or comment
String[]	getLikes()	gets all likes from certain person within a timeframe
int	countLikes()	gets the number of likes for a comment or post
void	addEvent()	adds new event
pair<string,long>[]	getEvent()	gets all events within timeframe
void	acceptEvent()	accepts notification of an event
void	declineEvent()	declines notification of an event
void	addKey()	adds a public key to the DB
PublicKey[]	getKey()	gets the public key for a username, or all which are stored)
string	getName()	gets a username for the given public key
void	addCategory()	adds a new category to the DB
void	addToCategory()	adds a user to a category

Table 24.6: Database

return	function	description
N/A	GUI()	Constructs a GUI
void	run()	continually updates the GUI from the DB
void	close()	kills the GUIServer thread
boolean	isRunning()	returns true if the GUIServer is running, false otherwise

Table 24.7: GUI

return	function	description
N/A	Message()	Constructs a message with given data
Message	parse()	(static) parses the string representation of a message into a message
String	toString()	creates a string representation of the message
String	getCmd()	returns the type of message
String	getContent()	returns the content of the message
String	getSig()	returns the RSA signature on the message
long	getTimestamp()	returns the timestamp on the message

Table 24.8: Message

return	function	description
N/A	Pair()	Constructs a pair with given data
A	first()	returns the first value passed to the constructor
B	second()	returns the second value passed to the constructor

Table 24.9: Pair<A, B>

24.2 Class Diagram

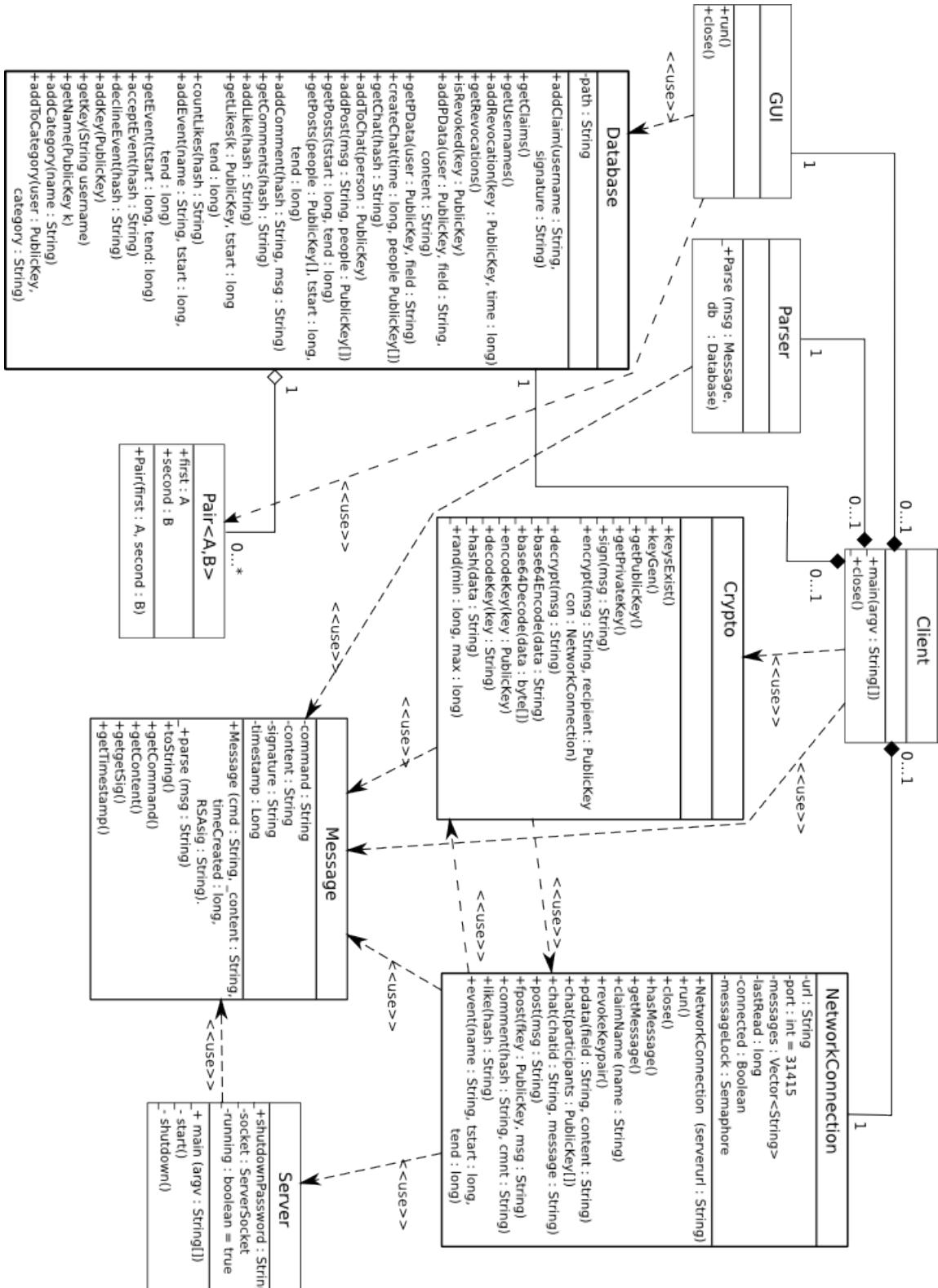


Figure 24.1: UML Class Diagram

Chapter 25

Pseudocode

25.1 Server

```
static void main () {
    startGUItread ()
    startServer ()
}

static void start () {
    socket = new ServerSocket (port)
    while (running) {
        incoming = socket . accept ()
        t = new Thread (new Session (incoming))
        t . start ()
    }

    shutdown ()
}
```

25.2 Client

```
static void main () {
    NetworkConnection connection      = new NetworkConnection ("server . tld ")
    Thread                  networkThread = new Thread (connection)
    Database                db          = new Database ("./ db")
```

```

GUI           gui           = new GUI(db, connection)
Thread        guiThread     = new Thread(gui)

if (!Crypto.keysExist())
    Crypto.keyGen()

networkThread.start()
guiThread.start()

while (gui.isRunning())
    while (connection.hasMessage())
        Parser.parse(Crypto.decrypt(connection.getMessage()), db)
}

```

25.3 Crypto

```

keyGen () {
    keypair = generateRSAkeypair()
    pw      = GUI.getUserInputString()
    filesystem.write("keypair", Crypto.aes(pw, keypair))
}

static String sign (String msg) {
    byte[] sig = SHA1RSAsign(msg.getBytes("UTF-8"), Crypto.getPrivateKey())
    return Crypto.Base64Encode(sig)
}

static String encrypt(String cmd, String text, PublicKey recipient,
                      NetworkConnection connection) {
    Message msg = new Message(cmd, text, connection.getTime() + Crypto.rand(0, 50),
                              Crypto.sign(text))

    // encrypt with random AES key with random initialization vectors
    byte[] iv = new byte[16]
    byte[] aeskey = new byte[16]

    fillWithRandomData(iv);
    fillWithRandomData(aeskey);
}

```

```

byte[] aesCipherText = aes(aeskey, iv, msg.toString().getBytes("UTF-8"))

// encrypt AES key with RSA
byte[] encryptedAESKey = rsa(Crypto.getPrivateKey(), aeskey)

// "iv\RSA encrypted AES key\ciper text"
return Base64Encode(iv) + "\\" + Base64Encode(encryptedAESKey) +
       "\\" + Base64Encode(aesCipherText)
}

static Message decrypt(String msg) {
    // handle claim messages (which are the only plaintext in the system)
    if (msg.substring(0, 2).equals("c "))
        return Message.parse(Base64Decode(msg.substring(2)))

    // handle encrypted messages
    String[] tokens = new String[3]
    tokens = tokenize("msg", "\\")

    byte[] iv           = Base64Decode(tokens[0])
    byte[] cipheredKey = Base64Decode(tokens[1])
    byte[] cipherText  = Base64Decode(tokens[2])

    // decrypt AES key
    byte[] aesKey = rsaDecrypt(cipheredKey, getPrivateKey())

    // decrypt AES Ciphertext
    aes.init(Cipher.DECRYPT_MODE, aesKeySpec, IVSpec)
    byte[] messagePlaintext = aesDecrypt(cipherText, aesKey, iv)

    return Message.parse(messagePlaintext)
}

```

25.4 Database

Most database functions are just going to construct parameterized SQL queries to be sent to the database from passed parameter values. The exceptions which include significant computing are

listed here:

```

void addKey (PublicKey k) {
    for each row r in table message_claim
        if (Crypto.verifySig(r.signature, k))
            addFriend(new Friend(k, r.username))
}

PublicKey[] getKey (String username) {
    PublicKey[] keys
    for each row r in table user
        if (r.username == username)
            keys.add(r.public_key)
    return keys
}

void addToCategory (Friend f, String category) {
    for each row r in table wall_post
        if (r.permission_to includes category)
            sendMessage(r, f)
}

```

25.5 Network Connection

The vast majority of messages here merely construct the appropriate message from the parameters and pass it to serverCmd()

```

void main (String _url) {
    url = _url
    messages = new Vector<String>()
    messageLock = new Semaphore(1)
    connected = true

    File lastReadFile = new File("./db/lastread")
    lastRead = Long.parseLong(lastReadFile.readLine())
}

void run () {
    while(running) {

```

```

        sleep (delay)
        downloadNewMessages()
    }

}

String[] serverCmd (String cmd) {
    Socket s;
    BufferedReader in;
    PrintWriter out;

    // connect
    s = new Socket (new Proxy.Proxy.Type.SOCKS, new InetSocketAddress ("localhost", 9050))
    s.connect (new InetSocketAddress (url, port))
    in = new BufferedReader (new InputStreamReader (s.getInputStream ()))
    out = new PrintWriter (s.getOutputStream (), true)

    // send command
    out.println (cmd);
    out.flush ();

    // receive output of server
    Vector<String> output = new Vector<String> ();
    String line = null;
    do {
        line = in.readLine ();
        if (line != null)
            output.add (line);
    } while (line != null);
}

```

25.6 Parser

```

void parse (String msg, Database db) {
    Message m = Message.parse (msg)
    if (m.cmd == "PDATA") {
        String [] tokens = tokenize (msg.content, ":")

```

```
    db.addPData(tokens[0], tokens[1])
} else if (m.cmd == "REVOKE") {
    PublicKey key
    for row r in table users
        if Crypto.verifySig(r.public_key, m.signature)
            key = r.public_key
    db.addRevocation(key)
} else if {
    etc...
}
```

Chapter 26

Database

26.1 Database design description

Note the difference between 'main user' and 'user'. Main user refers to the user who owns the local database. 'User' or 'other user' refers to other users, usually the relations of the main user.

26.1.1 user table

This table stores user details, which includes the main user's own details and its relations. As the user makes a new relation with another user, its details will be stored in this table. Every user has their own public key which uniquely identifies their accounts which also be stored in this table.

26.1.2 user, is_in_category, category table

With the category table, the user can create new categories to group his relations. As it is possible for many users to belong in many categories, the *is_in_category* table is needed to identify which set of users belong in the categories.

26.1.3 user, is_invited, events table

These tables suggest that users can create events. One particular feature regarding these tables that on the *is_invited table*, where the user (the main one) can invite anyone individually from the relations list or as a group from the category list. However, there will be no tuples added under this table when another user posts the event. Reason being is that the main user is not allowed to see who the list of other users invited in the event which was not created by the main user.

When the main user creates an event, he invites other people, either from the user table or from the category table or both. Once the invitation is sent out to those users, the users can either accept or reject the invitation. Using the *decision* attribute from the *is_invited* table, if decision has not been made, it will be NULL. If user accepts the invitation, it will be 1 for true. If rejected, it will be 0 for false.

26.1.4 user, allowed_to, wall_post table

When users create post, its data will be inserted into the *wall_post* table. The attribute *from* refers to the user who has created the post, whilst the attribute *to* refers to the user who is referred or mentioned in this post. The main user can also choose to allow a set of his relations to view his post. Using the *allowed_to* table, similar as the *is_invited* table, the main user can select his relations either individually or through categories or both. If the post is created by another user, no tuples will be inserted into the *allowed_to* table.

26.1.5 user, has_like, wall_post table

Users can like any posts that appear in his main wall or personal wall. When a post is liked, a new tuple is created in the *has_like* table to identify who liked the post, which post is liked, and the time the post is liked. These likes are counted and displayed in the GUI showing how many users have liked this post.

26.1.6 user, has_like, has_comment table

Other than liking posts, users can like individual comments as well. Same feature as liking the post by this time, data is inserted into the attribute *comment_id* from the *has_like* table to show which particular comment has been liked by this user.

26.1.7 user, has_comment, wall_post table

Users can comment on posts. When post is commented on, a new tuple will be added into the *has_comment* table on information like the content of the comment, which post has been commented on, who commented on the post, and the time of comment.

26.1.8 user, has_comment table

Users can also comment on comments itself. This will create and indent on the GUI to suggest that the parent comment has a child comment. When a comment is commented upon, the attribute *comment_comment_id* will insert the parent comment_id which shows the relation of two comments, one parent and the other being the child.

26.1.9 user, is_in_message, private_message table

Another functionality found in Turtlenet is the user is able to send private messages to users. When a private message is created by the main user, a new tuple is added into the *private_message* table. The user then has the option to add other user(s) into the conversation. When done so, a tuple or tuples, depending on the number of users he has added onto the conversation, are added into the *is_in_message* table. This inserts the information such as the time of when the user has been added into the conversation, the user's ID and message ID. The *private_message* table on the other hand stores data such as the content of the message and the time for which this whole conversation was created.

26.1.10 message_claim table

This table stores all CLAIM messages which cannot be matched with a public key. When a new key is entered we search for the CLAIM message, erase it, and add a new entry to the user table.

26.1.11 key_revoke table

This stores key revocation messages. If a user suspects that their private key has been compromised then they can send a message informing their relations of this. Once a key revocation message is sent all content posted after the given time and signed with the corresponding private key is marked as untrusted.

26.1.12 login_logout_log table

This table simply tracks the login and logout activities of the main user. When a user logs in and out, a new tuple will be inserted into this table.

26.2 Logical table design version 1.0

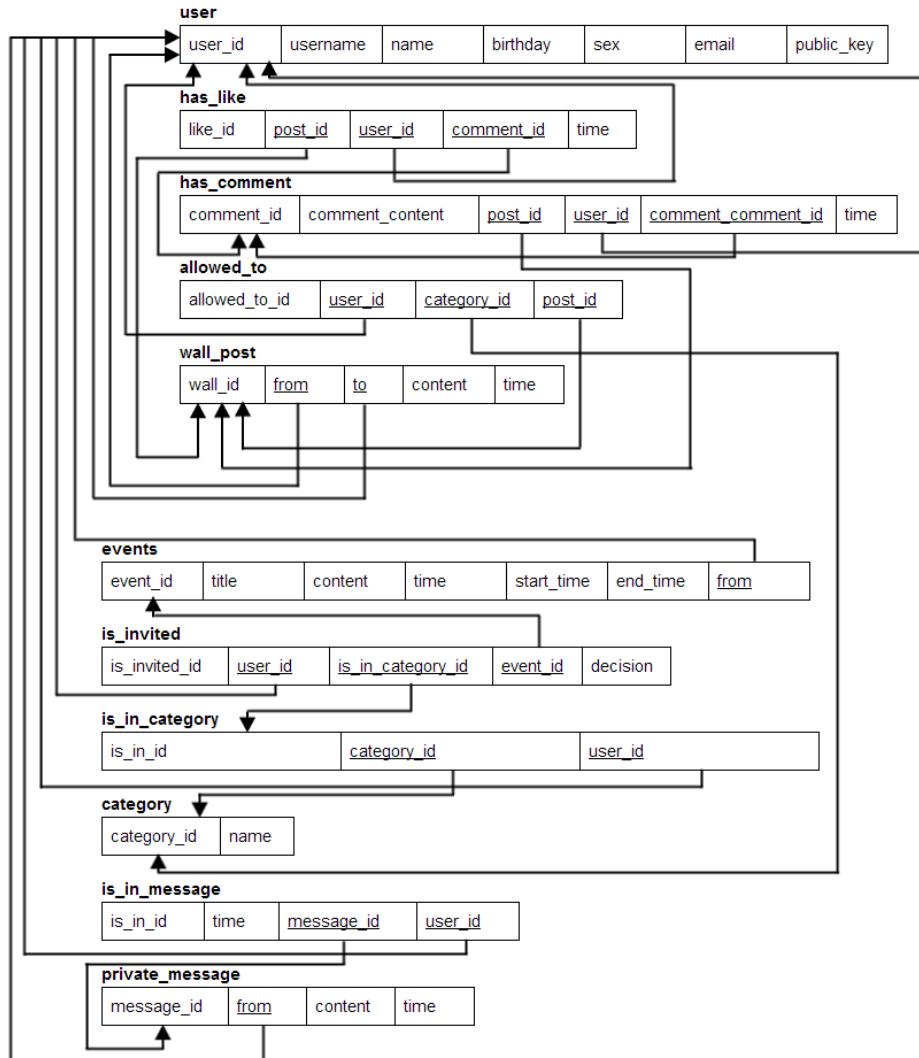


Figure 26.1: Logical design of database part 1

login_logout_log		
log_id	login_time	logout_time
message_claim		
username	signature	
key_revoke		
revoke_id	signature	time

Figure 26.2: Logical design of database part 2

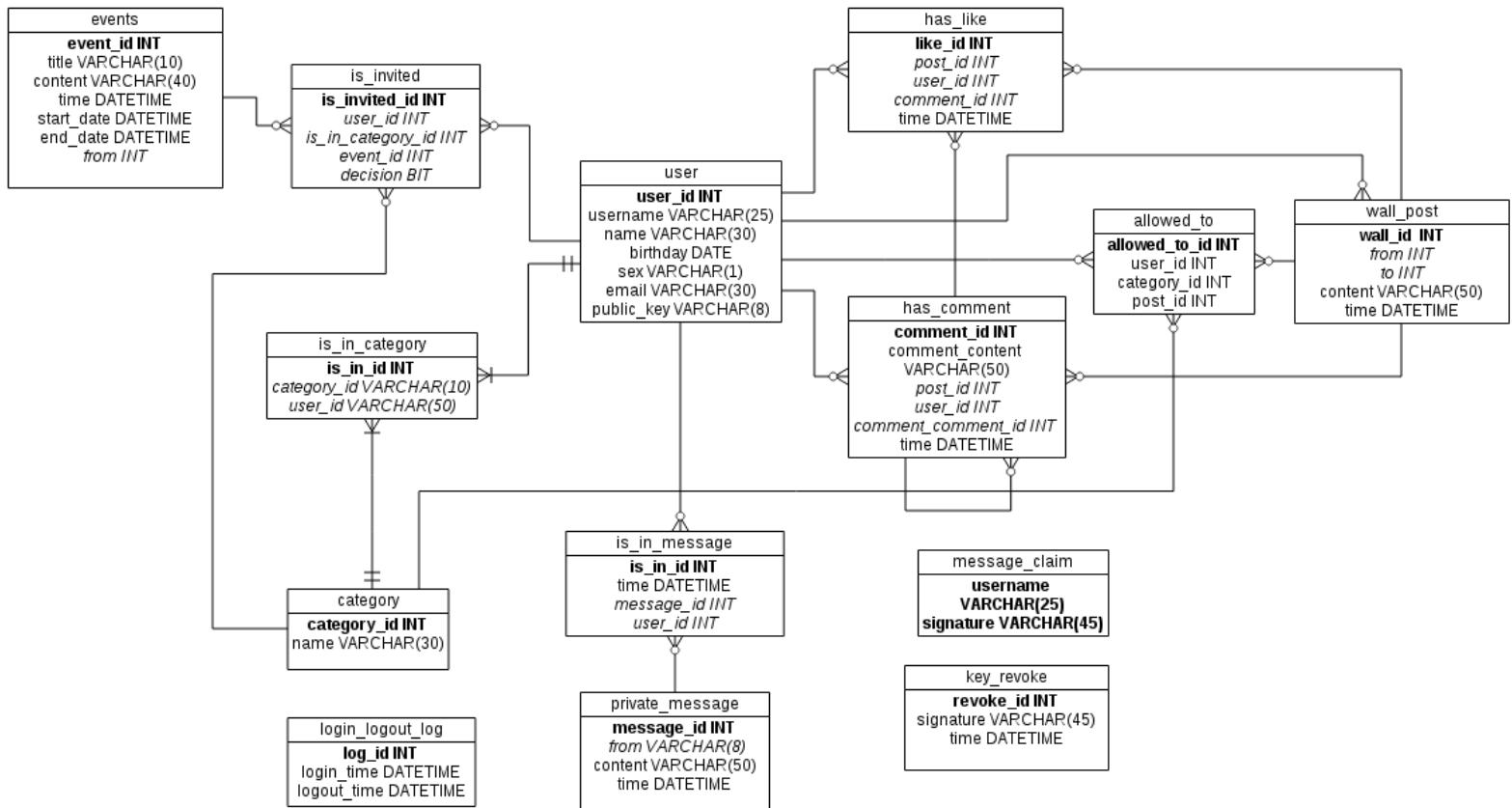


Figure 26.3: ER diagram version 1.0

Chapter 27

Transaction details

The table below shows the transaction details of each function which will be found in the program. There are four types of transactions for databases which are insert, read, update and delete.

Insertion is done when new data is added into a NULL attribute. Read on the other hand, is to view information from selected table(s) and its attribute(s). Similar as insertion but update is conducted when data already exists in the particular attribute. This basically removes previous data and add a new one. Lastly, delete, as it is self explanatory, deletes the whole tuple from the database. However this is usually avoided in database norms.

Function	Table(s) involved	Transaction(s)
addClaim()	message_claim	Insert
getClaims	message_claim	Read
getUsernames	user	Read
addRevocation	key_revoke	Insert
getRevocations	key_revoke	Read
isRevoked()	key_revoke	Read
addPData()	user	Update
getPData()	user	Read
createChat()	private_message	Insert
getChat()	private_message, is_in_message	Read
addToChat()	is_in_message	Insert
addPost()	wall_post	Insert
getPosts()	wall_post	Read

Function	Table(s) involved	Transaction(s)
addComment()	has_comment, wall_post	Insert, Read
getComments()	has_comment, wall_post	Read
addLike()	has_like, wall_post, has_comment	Insert, Read
getLikes()	has_like, wall_post, has_comment, user	Read
countLikes()	has_like, wall_post, has_comment	Read
addEvent()	events	Insert
getEvent()	events	Read
acceptEvent()	events	Update
declineEvent()	events	Update
addKey()	message_claim, user	Read, Delete, Insert
getKey()	user	Read
getName()	user	Read
addCategory()	category	Insert
addToCategory()	category, is_in_category	Read, Insert

Chapter 28

User Interfaces

28.1 Interface Research

As a social network, the user interface design is of high importance, as a lot of users of the program will have little core system knowledge, and rely entirely on the user interface. As a result we have looked at a variety of options into designing which will be the best for the project.

28.1.1 Swing

Swing is the primary Java GUI toolkit, providing a basic standpoint for entry level interface designing. Introduced back in 1996, Swing was designed to be an interface style that required minimal changes to the applications code, providing the user with a pluggable look and feel mechanism. It has been apart of the standard java library for over a decade, which, as I will now explain, may not be to our benefit.

Swing, whilst an excellent language to begin with, and write simple applications in, is quite dated. As our group advisor put it when inquiring about what we would be coding the user interface in:

"You should avoid Swing to prevent it looking like it was done in the nineties." - Sebastian Coope

Sebastian is not wrong either, as Swing does a very plain feel to it. This figure shows an old instant messaging system written with Swing by one of our team members. As you can see it is unlikely to appeal to the mass market with such visually plain appearance. This makes Swing, unlikely to be our GUI toolkit of choice, despite some of our members experience with it.

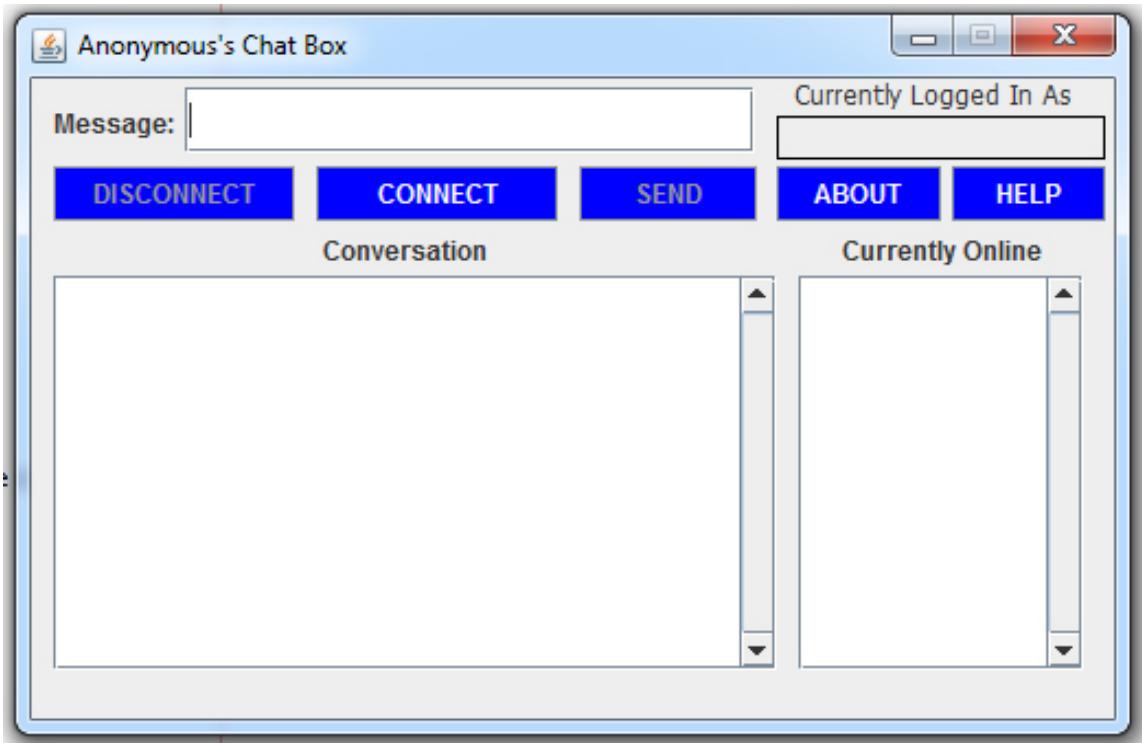


Figure 28.1: Swing Instant Messaging Application

28.1.2 Abstract Window Toolkit

Abstract Window Toolkit (otherwise known as AWT), was another choice given that we are programming in Java, and synchronicity between the two would be an advantage. Whilst AWT retained some advantages such as its style blending in with each operating system it runs on, it is even older than Swing being Java's original toolkit, making any GUI displayed via it look rather dated. None of the the current team has any proficiency with AWT however, and whilst it is possible to learn, there are still other options to consider that may provide the use with a more professional GUI build.

28.1.3 Standard Widget Toolkit

Standard Widget Toolkit (otherwise known as SWT), is one of the more promising candidates so far given its look and up-to-date support packages. The latest stable release of SWT was only last year, and is capable of producing programs with a modern and professionally built appearance, as shown in the figure.

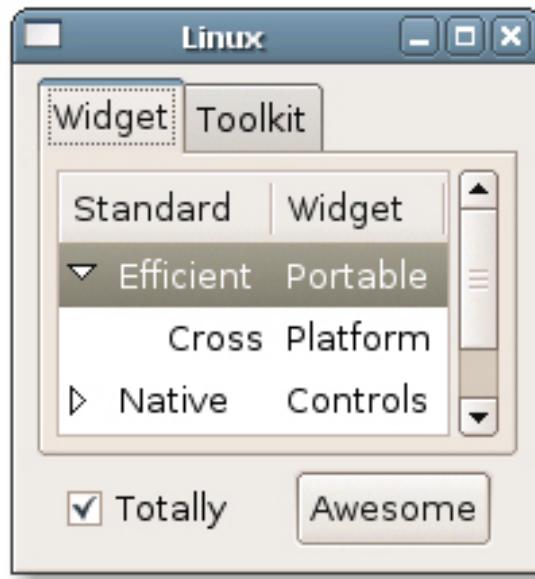


Figure 28.2: SWT Appearance Style

Unlike both Swing and AWT, SWT is not provided by Sun Microsystems as a part of the Java platform. It is now provided and maintained by the Eclipse Foundation, and provided as a part of their widely used Eclipse IDE, something a lot of the team is familiar with.

28.1.4 GWT

GWT allows you to create HTML/Javascript based user interfaces for Java applications running locally. The interface is programmed in Java and then GWT creates valid HTML/Javascript automatically. A web server is required in order for Javascript events to be sent to the Java application.

The user can then interact with the system by pointing their web browser at localhost. This has the benefit of being familiar to novice users as most modern computer interaction is done within a web browser.

Another advantage of using GWT is the ability to alter the appearance of web pages using CSS. This facilitates the creation of a modern, attractive user interface that integrates nicely with current operating systems and software.

28.1.5 Javascript

It is possible to create the entire client application in Javascript and use a HTML/Javascript GUI. This approach removes the need for a local web server meaning the only software the user is required

to run is a modern web browser.

Another advantage would be tight integration between the logic and interface elements of the client application and no risk of errors caused by using multiple programming languages.

One disadvantage of this approach is the difficulty in implementing the required security measures and encryption in Javascript. This can be remedied by using a Javascript library such as the Forge project which implements many cryptography methods.

The main disadvantage is that in this approach the server operator has complete control of the client the user uses. This is unacceptable because we're assuming that the server operator is seeking to spy on the user.

28.2 GUI Design

28.2.1 Client Design

Arguably the most important GUI in the project is the client GUI, as this is what the standard user will be interacting with, a person whom we are assuming has no knowledge of any inner workings. All tests we perform on our system at a later stage will be through this client, as per such its design takes a high level of importance. Its for this reason we have chosen something common users will be more accustomed to: web pages.

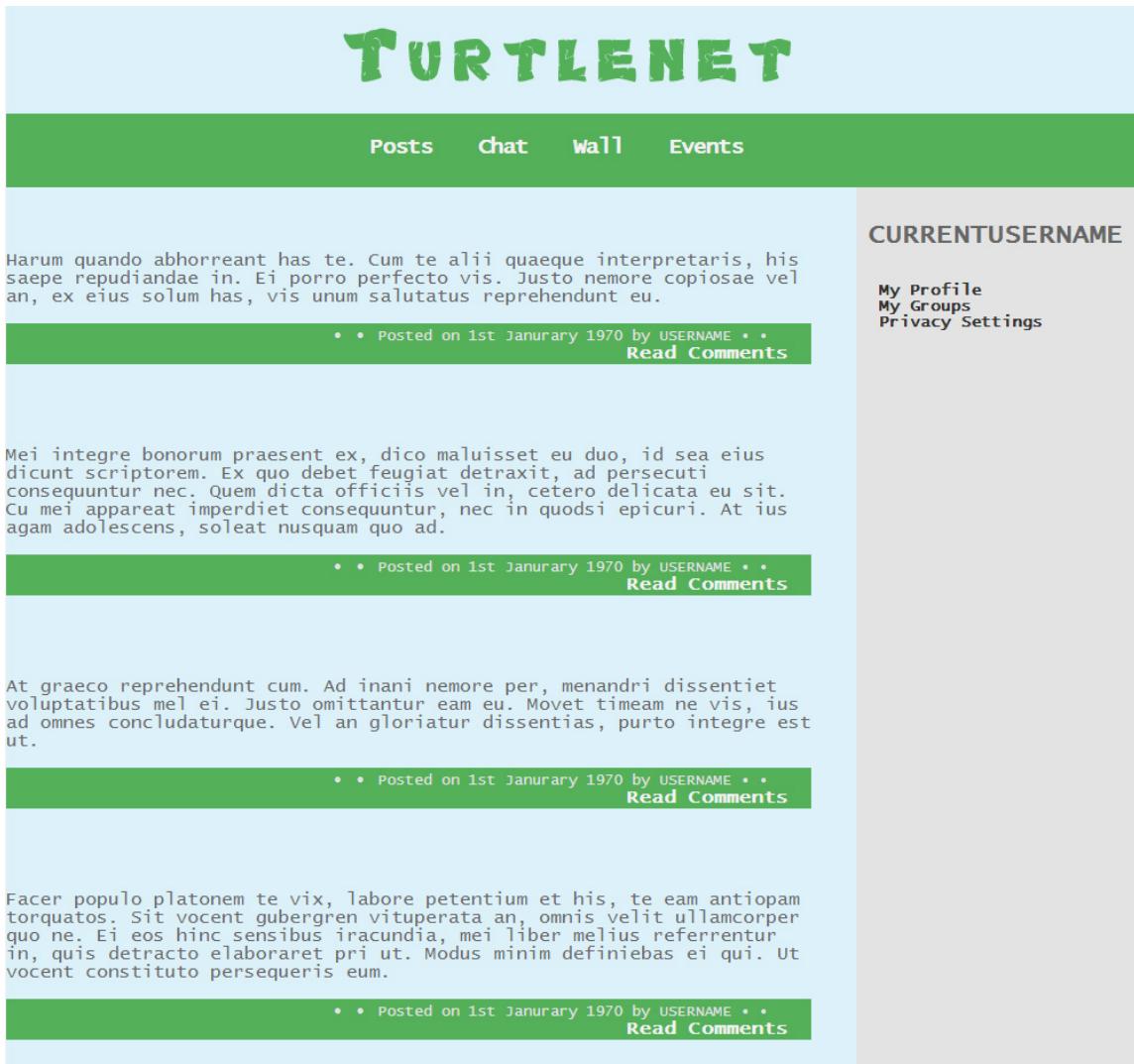


Figure 28.3: Client Design Image

Most users will be familiar with HTML and CSS page layouts, even if they do not know what HTML or CSS is. This will provide a certain level of comfort when it comes to using new applications and how to navigate between pages or tabs. Javascript would be used to pipe the data to the client program, but this is something the user would not interact with or see. It also provides the advantage of knowing nearly every operating system nowadays comes with a web browser natively meaning a HTML/CSS based GUI would likely be supported on nearly all platforms. For these reasons we have chosen to use GWT. A local web server has been decided as the best way forward, as it will

provide the best form of security from the server operators.

28.2.2 Server Design

Whilst not critically important, as it would only be operated by those with technical knowledge, is still an important aspect to consider. It needs to hold the system level settings and control mechanisms a server client would need, whilst not making them immediately and 'accidentally' accessible via the form of large obvious buttons. The easiest way of doing this is via a command input box beneath a chat log window to provide commands that way. It is also may be an idea to show server data such as memory usage on the operators end, as this data is completely accessible and non-intrusive to the client. The figure labelled 'Server Design Image' shows an example of how the server client may be completed. Pending on the features allowed in GWT, our method of choice, we will aim for it to retain a similar appearance.

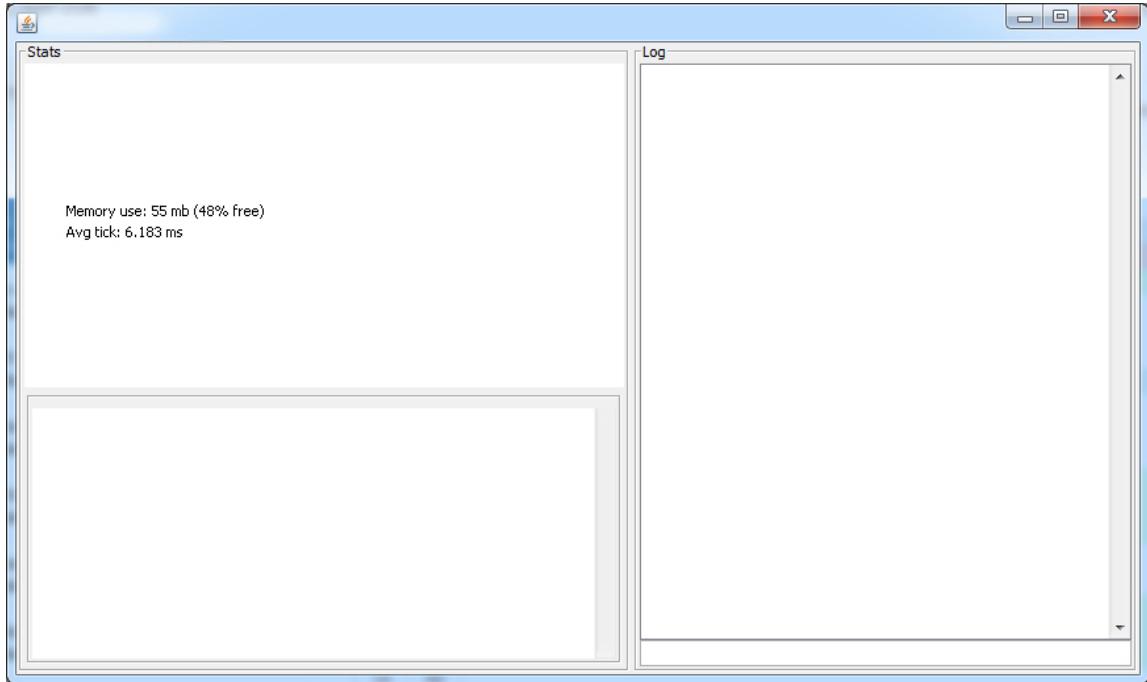


Figure 28.4: Server Design Image

28.3 Future Work

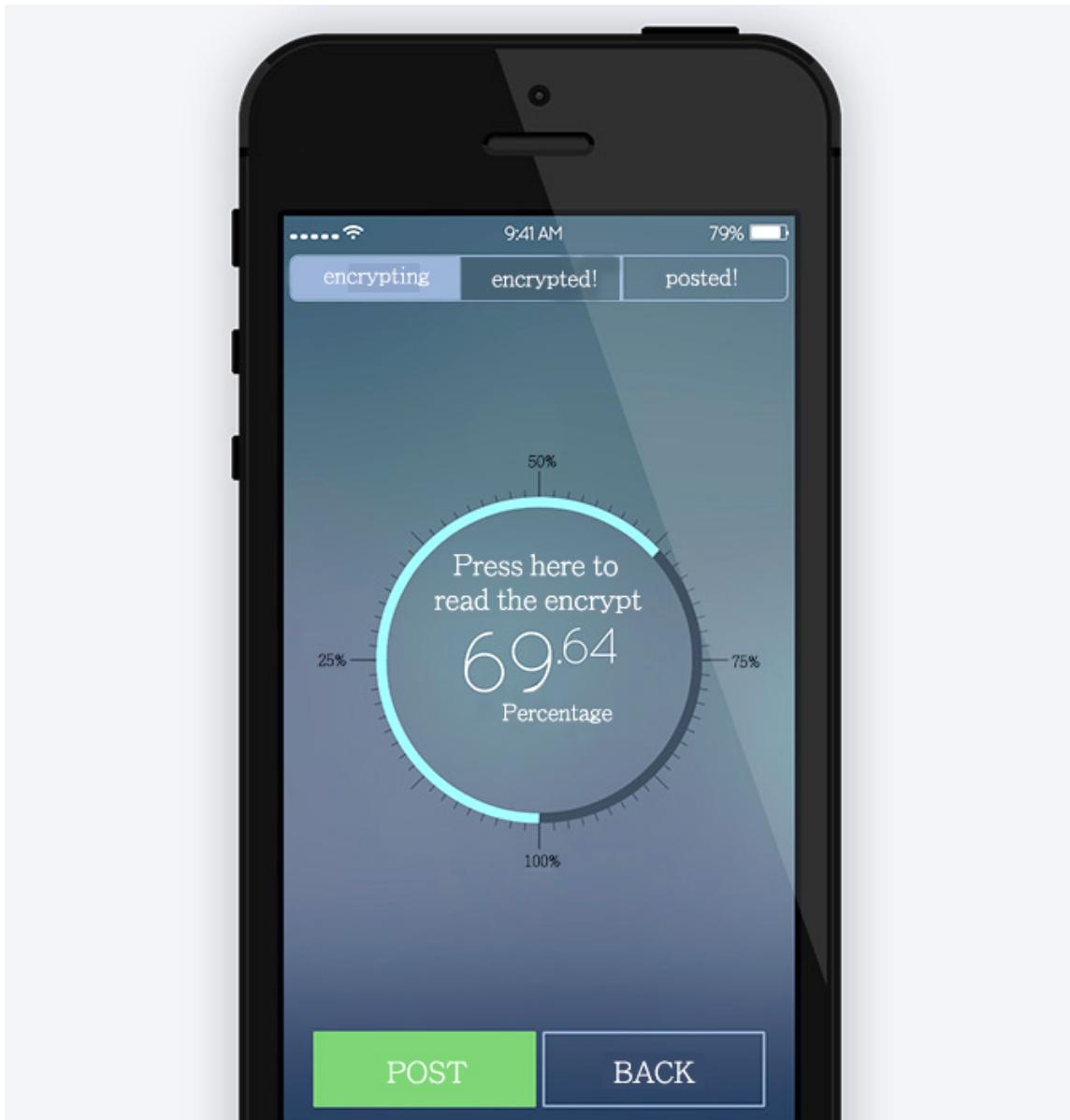


Figure 28.5: Mobile Sending Stage GUI

With some of the spare resources available during this phase, we were able to look into some future design work on the mobile front. One of our designers had some experience in this field of work

and offered to put some images together of what a mobile application version of our product could potentially look like.

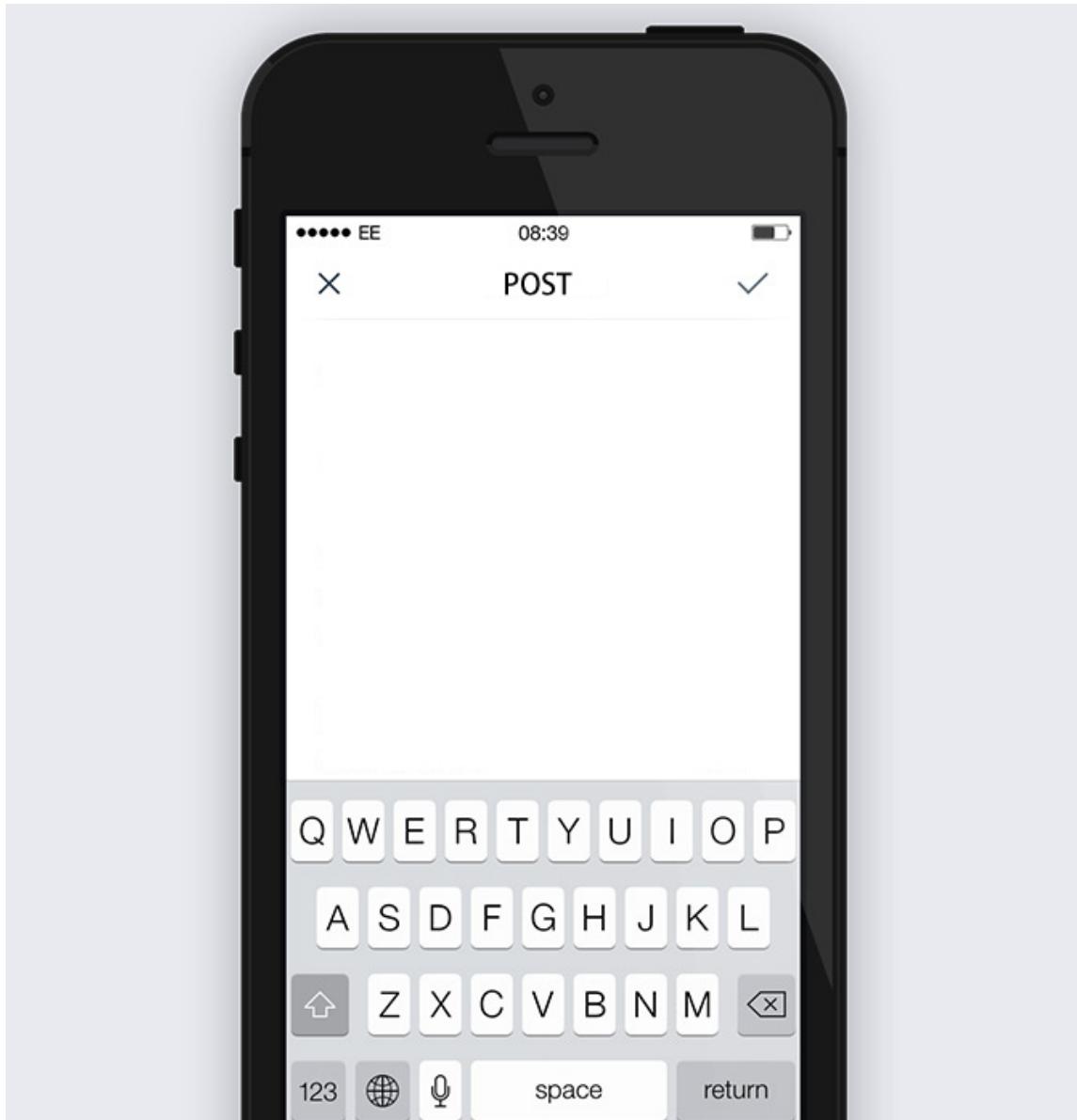


Figure 28.6: Mobile Post Stage GUI

The mobile interface data flow diagram shows how the application would flow between screens, giving an idea to the level of depth an application of this size might have.

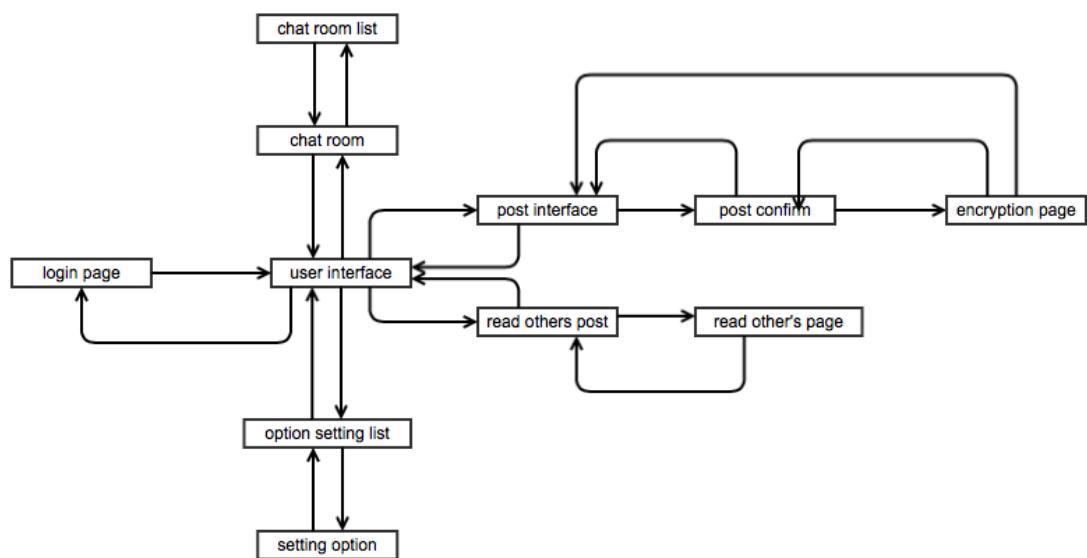


Figure 28.7: Mobile Data Flow Diagram

Chapter 29

Business Rules

In standard projects the business model can commonly outline certain validation practices for the program or project in the form of business rules or policies. Our project, and its fundamental idea, works a little different than most projects in terms of business, as per such we have only one business rule.

- To ensure the client never sends identifying data to the server or its operators.

This is to ensure that the privacy of communication is always within the hands of the client and user, as opposed to any who run the network. To violate this single rule would be going against both the company ideals, and the projects goals.

Chapter 30

Data Dictionary

Entity Name	Description	Aliases	Occurrence
User	A person who uses the program	Friend, contact, another user	A user can have contacts, posts, comments, events, convos, likes, or found in a category
Category	A type or a group of selected users which the main user classifies his contacts	Group	Some users may belong in a category
Category members	Users which belongs to category	users	Users can belong in a category
Claim	When a user gets another user's public key, the key is claimed.	Retrieving, getting	A user can have many claims
Revocations	A term for when the user marks a contact as malicious and blocks it	Block, mark	A user can revoke many users
Post	A set of words that has been written by user which displays on the wall when submitted	Wall post	Posts are displayed onto walls when submitted
Post visible to	Posts which are visible to a certain user	viewable post	Some posts are made viewable to certain users
Convo	A term for when two users or more talks to each other privately	Conversation, chat, private message	A user can initiate or be added into a convo
Convo keys	Public keys of those who are in the conversation	public key	Conversations may have users in it

Entity Name	Description	Aliases	Occurrence
Convo messages	Messages which are sent within the conversation	messages, chat	Conversations have messages
Comment	A term for text which is written directly for a specific post or comment which then displays on the wall		Comment appears on posts and parent comments
Event	A post that describes an activity, time and place		User can post events and invite users into it
Like	A response when someone finds a post favourable		Posts have likes

Attribute name	Description	Aliases	Found in entity	Occurrence	112
convоИD	Uniquely identifies conversations	primary key	convo	Every convo has an convo ID	
timeCreated	Time the convo is initiated		convo	Every convo has a creation time	
pk	Unique key identifier	primary key, key	convo keys	Uniquely identifies the row	
convоИD	Unique key identifier for conversations	primary key	convo keys	Every convo keys has a convo ID	
key	User's public key	public key	convo keys, user	Key is shown when user is in the conversation	
username	A human readable identifier for a user		user	Every user has a username	
knowName	A boolean figure to see if user knows the contact	know his name	user	If user knows the contact, value is 1, if not 0	
email	User's email		user	Every user has an email	
name	User's real name	name	user	Every user has a name	
gender	A term to suggest female or male	sex	user	Every user has a gender	
birthday	Date of birth	DOB, birthdate	user	Every user has a date of birth	
sig	Globally unique primary key	signature	user	Every user has a sig	

Attribute name	Description	Aliases	Found in entity	Occurrence
msgText	Written message in the comment	message, text	Comment	Every comment needs to have a message
senderKey	Owner's public key of the comment	public key	Comment	Every comment has a sender key
parent	The parent comment	comment, parent	Comment	Comment can be commented on
creationTime	Comment's creation time	time	Comment	Every comment has a creation time
msgText	Text message of the post	post message	post	Every post has a text message
time	Time of the post being created	creation time	post	Every post has a creation time
receiverKey	Public key of user who receives the post	public key	post	If post is directed to a user, it is to have a receiver key
sendersKey	Public key of user who creates the post	public key	post	Every post has a senders key
pk	The unique identifier for convo	primary key	convo messages	Every convo messages has its own identity key
convоИD	Refers to the convo ID	convo ID	convo messages	Every message belongs in a conversation
sendersKey	Refers to the owner of the message	public key, user	convo messages	Every message has a sender key
msgText	The content of the message	text	convo messages	Every message has a content in it
time	Time of when the message is sent	creation time	convo messages	Every message has a time when sent

Attribute name	Description	Aliases	Found in entity	Occurrence	114
sig	Globally unique primary key	signature	event	Signature is generated on every event created	
startTime	The start time of the event	start, time	event	Events have a start time otherwise not stated by user	
endTime	The end time of the event	end, time	event	Events have a end time otherwise not stated by user	
creatorKey	The public key of the owner of the event	public key	event	Every event has a creator key	
accepted	Users who accepted the invitation	users	event	When users accept the event invitation, it will be listed here	
name	The name of the event	Title, event name	event	Every event has a name	
creationTime	The time when event is created	creation time	event	Every event has a creation time	
pk	Unique identifier for like	primarfy key	like	Every like has a primary key	
likerKey	The public key of the person who likes the post	public key	like	Some posts have likes	
parent	The post of which the user has liked	post, wall post	like	Every like has a parent attribute	

Chapter 31

Glossary

AES - Symmetric encryption standard.

Category - We allow our users to create ‘Categories’, and place one or more users into one or more categories. These sets of users are used to speed up repetitive actions such as allowing all of your friends permission to view something, by instead allowing the user to allow the category ‘friends’ to view it.

Client - The program that will be used by users which connects to a turtlenet server.

FaceBook - A social networking website designed to make the world more open and to connect people together in a simple format.

Onion Routing - A manner of routing traffic in a network with the goal of obscuring from the recipient who the sender was. This is achieved by routing it through a number of intermediaries, none of which have access to both who sent the traffic, and the plaintext traffic.¹

Privacy - Personal information being known to only those whom you choose to inform of it.

Parameterized Query - A precompiled query lacking important information for the values of parts of it. These are used to protect against SQL injection and to provide a greater degree of abstraction from the database for the rest of the system.

QrCode - QR stands for Quick Response. Used to store data it is a form of 2D bar code, It was designed to be easy to read from low quality photographs.

Relation - Two users must know each others public keys in order to communicate. We say that two users who possess each others keys are in a ‘relation’. This is done because it is a situation we talk about often, and it helps to have a word for it.

RSA - An asymmetric encryption algorithm.

SocialNetwork - A website build around facilitating social interaction.

Server - A computer running the turtlenet server which allows clients to connect to it.

¹See http://en.wikipedia.org/wiki/Onion_routing for more information.

ServerOperator - The owners and engineers responsible for running Turtlenet servers.

Tor - An implementation of onion routing.

Part III

User Manual

User Manual - Turtlenet

Ballmer Peak

M. Chadwick, P. Duff, A. Senin, L. Thomas

May 9, 2014

Contents

1 General	3
1.1 System Overview	3
1.2 Contact	3
2 Getting Started	4
2.1 Getting started	4
2.2 System Requirements	4
2.3 Installing Turtlenet	4
2.4 Running Turtlenet	5
2.5 The Turtlenet Interface	5
2.6 Account Creation	6
3 Using the System	7
3.1 Creating an Account	7
3.2 Logging into Turtlenet	9
3.3 Navigating around the Turtlenet client	10
3.4 Logging out	10
3.5 Friends on Turtlenet	11
3.5.1 The 'Getting'	11
3.5.2 The 'Making'	12
3.5.3 Banding Together	12
3.6 Messages in Turtlenet	14
3.7 What's mine is mine - Personal Data	14
3.8 Personal Graffiti - your Turtlenet wall	16
4 Troubleshooting	18
4.1 Frequently Asked Questions	18
4.1.1 What does Turtlenet do?	18

CONTENTS	3
4.1.2 How many accounts can I have on Turtlenet?	18
4.1.3 I forgot my password. Can someone reset it for me?	19
4.1.4 Where is everything stored?	19
4.1.5 How big does this database get?	19
4.1.6 Why would someone want to build from source?	19
4.1.7 The Client does stuff I don't think it should do...	19
4.1.8 What do Server Moderators of Turtlenet do?	19
4.1.9 I want to mod Turtlenet. Can I have the source?	19
4.1.10 Why choose 'X' over the clearly superior 'Y'?	20
5 Side Notes	21
5.1 Java Database Connector	21

Chapter 1

General

1.1 System Overview

Turtlenet is a purpose-built, privacy oriented social network, which demands zero security or technical knowledge on behalf of its users. It allows communication between users securely, which can either be in the form of instant messaging, or creating posts on users walls.

What makes Turtlenet significant is even the service operators are unaware of who communicates with whom. It is designed from the ground up that they can never know this, even if they wanted to. This resolves a more common security issue that plagues modern social media networks, an issue Turtlenet has been created to not have.

1.2 Contact

Team contact information:

- p.duff@turtlenet.com
- l.thomas@turtlenet.com
- a.senin@turtlenet.com
- l.prince@turtlenet.com
- m.chadwick@turtlenet.com
- l.choi@turtlenet.com

Chapter 2

Getting Started

2.1 Getting started

Welcome to using Turtlenet! Through the use of Turtlenet, you will experience the ease of use and the practicality of communicating and socialising with your friends, family, business associates or anyone else that you know through a medium where your data is ensured to be protected. This user manual has been designed and written specifically to assist the users by providing detailed description of all the various uses of the program. Let's get started!

2.2 System Requirements

These are the minimum system requirements for Turtlenet:

- An internet connection
- Any OS with a JRE (version 1.6.x or higher)
- Any up-to-date browser

2.3 Installing Turtlenet

In order to install Turtlenet, you simply download ONE of the files from our website:

www.turtlenet.co.uk/downloads.html

Most users will want to get the version that is without 'TOR' as unless you know what that acronym stands for, you won't have it installed. It is an external piece of networking software which

adds another layer of security, hiding your IP address so people don't know where you currently are.

As the file is a Java Archive (JAR), you can put it in whatever folder you choose - Turtlenet doesn't mind. It will create the required files and folders when it is running so just pick a pleasant home for the download.

2.4 Running Turtlenet

Now you have the client on your computer, you will need to run it. People who are familiar in using Java may be able to work it out but this section is here for those who want to make sure that they are going to run it first time correctly and without frustration. Here is what you do:

1. Open Command Prompt (Windows) or your Terminal (*nix and OS X)
2. use 'cd' to get to where your Turtlenet client .jar file is. Windows users changing drive letters will need the '/D' parameter. e.g. 'cd /D D:
TurtlenetFolder
,
3. You will want to run the java command:
`'java -jar turtlenet.jar'`

If you managed to get to the downloaded client JAR file and ran that command, you should have the back end of the Turtlenet client running. All you need to do now is open your preferred browser, or one of the suggested browsers if you have more than one, and type '`localhost:3141`' into your URL bar.

If the browser did not complain about anything and just worked, you should see a Turtlenet banner. If so, you have your client running successfully!

2.5 The Turtlenet Interface

Turtlenet comes with a simple interface that has the main menu, which has the following sections:

- My Wall
- My Details
- Messages
- Friends
- Logout

2.6 Account Creation

The user is expected to create a new account when using Turtlenet for the first time. In order to create an account, enter a user name and a password, as well as repeating your password into the confirmation box. Once the user has created an account they will be logged into Turtlenet. From here onwards, the user can then add further profile details should they wish to. How to do so will be explained under the 'Using the System' section.

Chapter 3

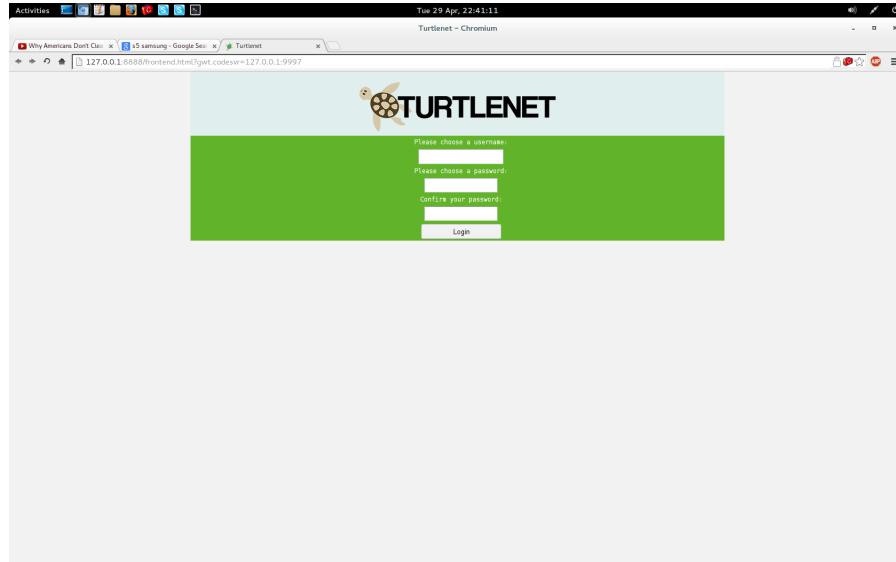
Using the System

This section extends upon the fundamentals mentioned in the Turtlenet (TN) general section.

3.1 Creating an Account

The 'General' chapter only briefly mentions creating an account so to make this section complete as a 'go-to' resource for users it will also be mentioned here too.

This is where your private communications begin.



This image shows the account creation page, which you should see when you run the client for the first time on your computer. From the top there are three text boxes:

- a Username box
- a Password box
- a Confirmation box

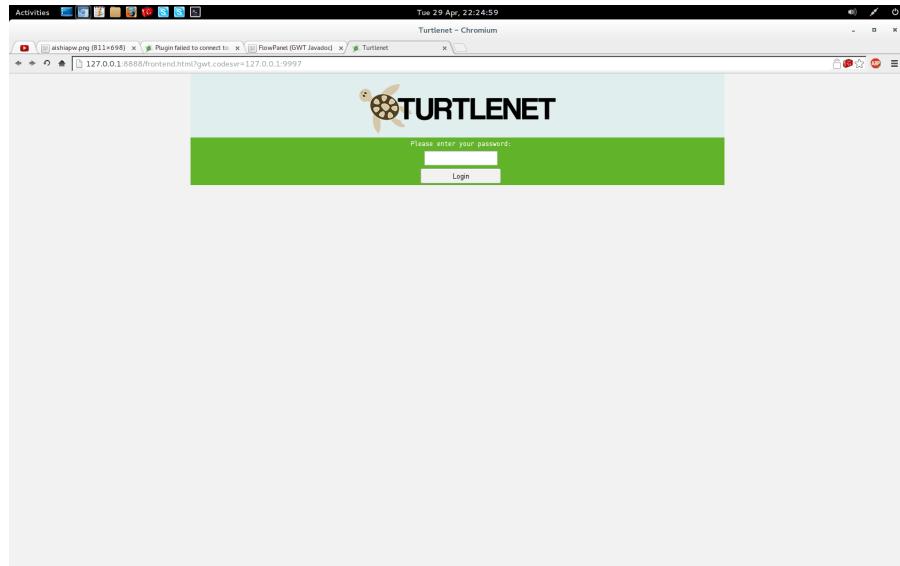
You fill in each of the fields with the required information which will be the following:

- The Username box should be filled in with your user name. This is what other users would call you when posting messages. This should be something that represents you, but should not link to you outside of Turtlenet. Simply, your Turtlenet user name should not be the same as any other user name you use on the internet. If the name can be linked to you then people are able to easily determine that you have a turtlenet account.
- Your password should be easy to remember but difficult for anyone else to guess. A good method for coming up with new passwords is to use four or five words, in a phrase. An example would be 'ThisIsTurtlenetzPassword'. This is better and easier to remember than what is usually suggested which is a shorter password with numbers in them: 'P@ssw0rd'. Of course, it depends on who is remembering the password so choose your own method if either option mentioned feels uncomfortable for you.
- The Confirmation box is where you type the password you defined in the previous box. Because of this, they should match, and must if the account creation is to be successful. The easiest way of thinking about this box is that it is giving you the practice of inputting your password while it is still fresh in your mind, to help you remember for later on.

By filling in these text boxes with the kind of information mentioned in this section, you can then click the button underneath these boxes to create your account. If successful you will be automatically logged in.

3.2 Logging into Turtlenet

Logging into the Turtlenet client is as simple as using the password that you had used to create your account.



The screen shot shows the initial page you might see once you have created an account. Enter your password into the white text box above the 'Login' button and if the password is correct, you would have logged in.

3.3 Navigating around the Turtlenet client

Getting around the client's various areas is important in order to make the most of the functionality provided by Turtlenet. This is why all of the main segments are provided as buttons at the top of the interface:



The image shows that there are several main sections to the client - The wall, the user's details, messages between the user and other people, friends that the user has linked with and finally the function to logout. Click the corresponding button to get to the area you wish to view. The following sections will go through each section from right to left.

3.4 Logging out

For when you decide that you want to leave the safety of Turtlenet and work on other things, or you simply need to be away for a while and want to be sure that no one is using your account, you will want to log off. It is as painless as clicking the 'log off' button found at the top right of every page. Doing so will take you to the login screen (the one with just the password box and login button). Of course, we wish you good fortune until you come and join us again at Turtlenet.

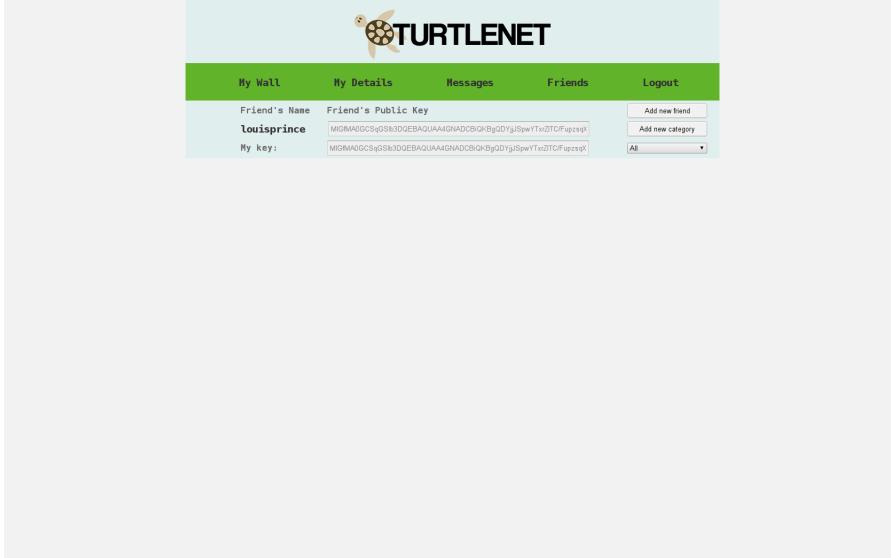
3.5 Friends on Turtlenet

Part of the philosophy of Turtlenet is to encrypt the messages that you send so that only the intended recipients can read them with any understanding. These people are known as your 'friends' on Turtlenet. In order to make any use of Turtlenet you need to add friends. You do this by exchanging 'public keys' with another user. Turtlenet uses Asymmetric relationships - this means that you may have some people as friends but they might not have you as a friend. Therefore you might understand what people have typed but they might not be reciprocated. If this doesn't make sense at the moment, the following sections will help.

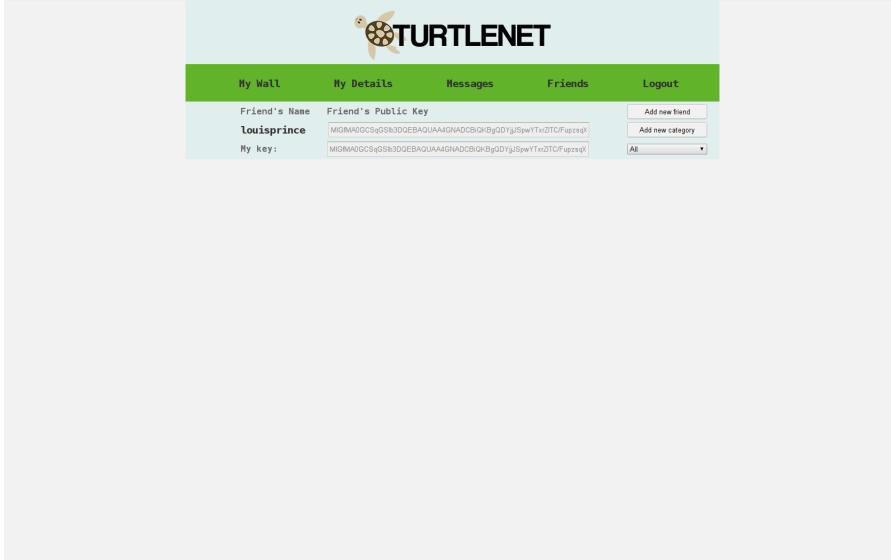
3.5.1 The 'Getting'

In order to get public keys from other users, they need to pass the information to you. The keys can be transferred in any manner, they are not remotely private and painting your public key on the side of your house would not diminish security.

Once you have the public key off of your friend, you will want to proceed to the 'friends' section of the Turtlenet client, by clicking the button near the top which has 'Friends' written upon it. You should either see the following or something to it's effect:



As you can see, there is 'My Key' which will be used by you to allow others to send you messages but that will be explained in the next section. For now, you want to click the 'Add new friend' button located to the right of the screen. This will bring you to a screen with a long input box which asks for the key of who will become your friend. You enter the long line of letters and numbers that you were given by your friend into the input box. Once you have the other person added, you should see something similar to this:



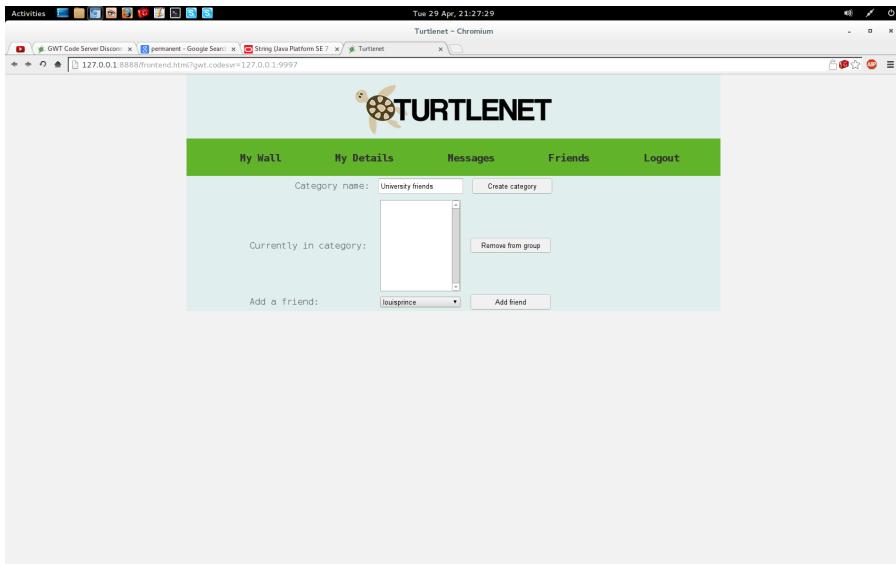
In the image above, the current user has added themselves to their friends list. Simply repeat the process as it takes you back to the main section for the friends tab.

3.5.2 The 'Making'

By getting other people's keys you can send messages to them but for people to send anything back that you can read, they would need to have your key as well. All you do in order to help others add you is to send the letters and numbers in the text box next to 'My key' and get the other user to follow the steps in the above section 'The 'Getting'.'

3.5.3 Banding Together

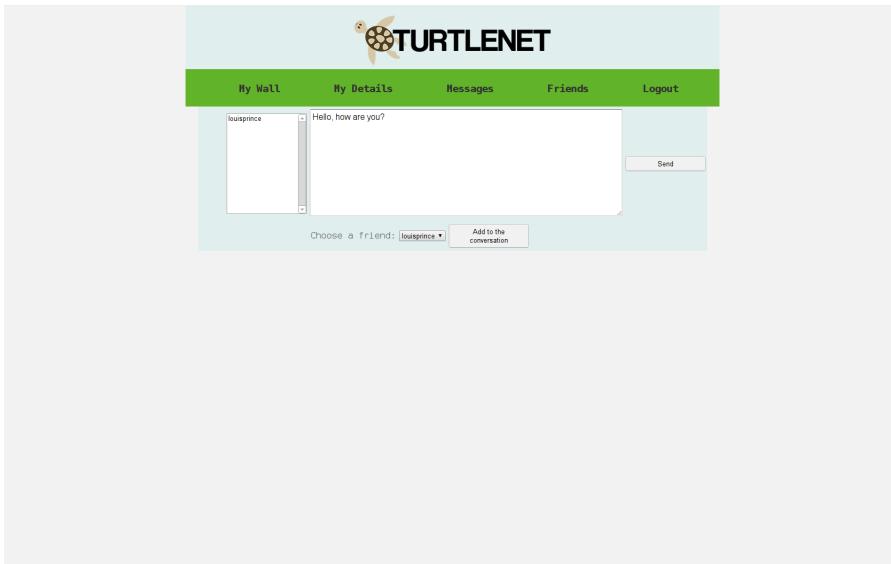
In Turtlenet you can associate other users with categories, custom made by you. This is useful if you want to send the same message to a number of people. To do this, whilst you are in the friends section of the client, click the 'Create category' button on the right. It should take you to this screen:



You will give your category a name so it hints to the kind of users you have in them together by typing the group name in the top text box. Click 'Create category' once you have finished the naming procedure. You are then able to add any members you wish whose keys you have attached to your account. This is done in the drop-down menu at the bottom of the interface and then clicking the 'Add friend' button next to said menu. If you no longer want a particular user in the group any more, select their user name in the large box in the middle and click the button to the side which says 'Remove from group'.

3.6 Messages in Turtlenet

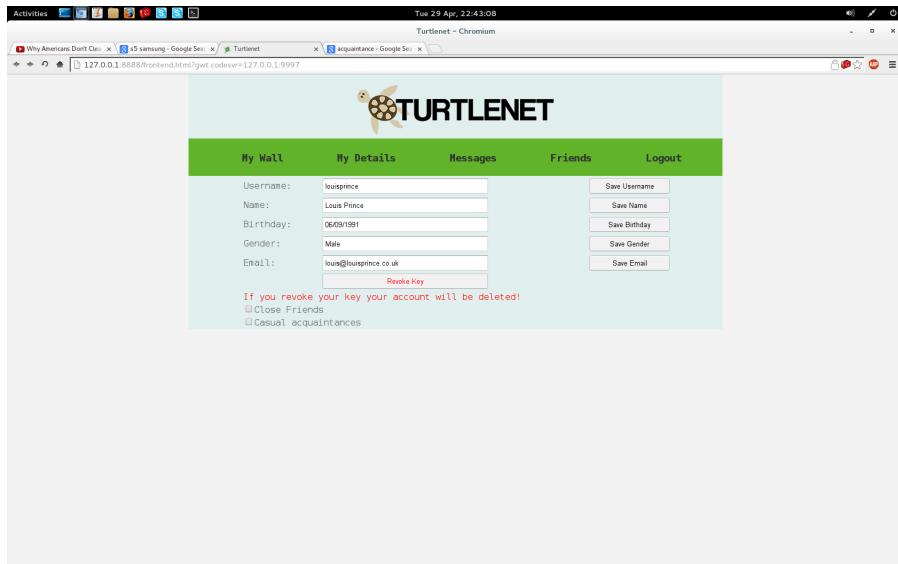
Messages can be sent to singular users or they can be sent to categories of users created by the current user of the client. Below is an example of what you may find in the messages section:



- the box at the left hand side is for your available recipients - our example user only has himself at the moment. This will fill up over time when you add public keys from other users.
- The larger of the two boxes is where you type the content of your message. There is no size limit.
- The Send button on the right finalises the message and sends it to the recipient to read. You cannot edit your message once you have sent it so be sure to re-read what has been typed to avoid any unfortunate errors!

3.7 What's mine is mine - Personal Data

When using Turtlenet, personal data is just that - personal. Similar to all of the messages and posts you make, your personal data is also encrypted and made secure so that the server moderators have no access to them. Here is a view at what you could see when entering the 'My Details' section of the client:



The image shows the only personal information that you may store using the Turtlenet client. Note that the only piece of information here that is important is the user name - all other fields are optional and at the user's discretion to fill in or not. Each button to the right saves what is currently in the associated field at the time of clicking, so you will need to save again if you edit after a save.

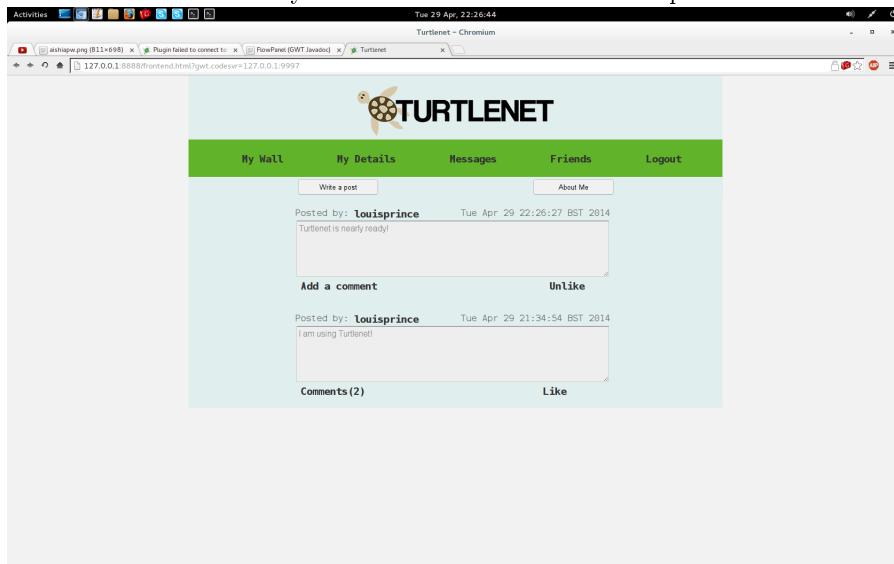
Below these fields is a list of categories you have created, check the box next to a category and members of it will be able to see your personal data. Unchecking the box hides any futures changes in it from them.

A note about revoking your key: This means that you mark your key as never again to be trusted, and so messages from it are ignored. **Do not click unless you wish to erase your Turtlenet presence.** After a revocation, another key is made for you to use, which means that any other users that had your key will need to be informed that you have changed and you will need to give them your new key if you wish to continue getting messages and posts from them.

3.8 Personal Graffiti - your Turtlenet wall

Your wall is a central social hub for many users of Turtlenet. It is a collection of messages aimed at the user, who may be off-line at the time. This section is for the functionality of the wall.

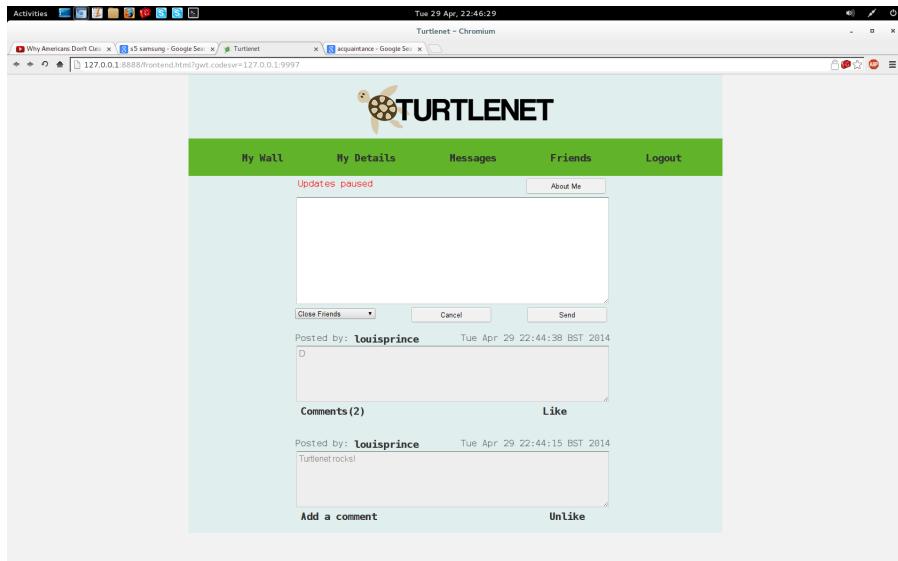
In Turtlenet a post is the generic way of talking about a message being left for another user - think of it similar to a sticky note on a cork board. An example of a wall is below:



The image outlines a couple of posts being made by the example user. Before posting is explained, this manual will explain the other elements in view:

- The 'About me' button allows a user to see an overview of their personal data. This allows a user quick access to their key, which could be sent to another user.
- You can 'like' posts to show enjoyment, appreciation or agreement with what another user has posted. This is done by simply clicking the 'like' that is found underneath the target post. Should your political views change for example, you can unlike any currently liked post in the same manner - clicking the 'unlike' that will be found in the same place under the target post.
- Commenting on a post is also possible with the Turtlenet client. Simply click the 'Add a comment' phrase underneath the target post and a large input box will appear beneath. Simply type your 'two cents' then click the 'Post comment' button under the input box. If you decide not to insert an interjection then you may click the Cancel button to remove the box and not attach your comment to the post.

Posting is as simple as clicking the 'Write a post' button near the top, which will bring a couple of new elements into the client:



As the above image shows, there are a couple of new buttons and a large text box that appears onto the Turtlenet client. First it is easiest to define a target for the post, which is done by clicking the drop down menu below the input box on the left side. The user is able to choose from categories that have been created, sending the post to multiple users. Once decided, type the content of the post into the large input box. Once finished, click the 'Send' button below the input box on the right side. If you wish to stop making a post, click the Cancel button in the middle, underneath the input box.

Chapter 4

Troubleshooting

4.1 Frequently Asked Questions

This is the section which should hopefully answer most of the questions that most users might have about the system. Sending emails to one of the addresses in the contact section in the beginning of the user manual may help you get your answer but it is best if you continue looking for an answer whilst you wait for an official reply.

4.1.1 What does Turtlenet do?

Think of Turtlenet in a similar manner to any other social network commonly in use. It allows users to communicate with each other and allowing other people to voice their opinions on what others have written. At the moment it is text based, meaning you can't attach images and video to it when you post or comment. You can however send links to such content to each other. That is a convenient enough work around for the time being as it means that no one is having to download an encrypted video but are never able to view it as they do not have the key to unlock the data. I think everyone will appreciate not having to download hundreds of copies of current top 40 each week.

4.1.2 How many accounts can I have on Turtlenet?

You should only need one, but we don't preclude you from having more. If you merely wish to separate the content people can see then categories are a better solution, and future versions of Turtlenet will allow the sharing of different personal information with different groups. If you simply must have multiple accounts though, there's nothing stopping you. Just launch the client in a different directory.

4.1.3 I forgot my password. Can someone reset it for me?

The short answer is no. Turtlenet was designed so that no one but the user had any access to their account. As a result, if you lose your password we are unable to recover anything in the account. The only thing you can do is simply to create another. Feel safe in the knowledge that everything is encrypted on your old account so at least no one can access what was lost except those people you already shared it with.

4.1.4 Where is everything stored?

Information is stored on your computer, laptop or whatever else it is that uses the Turtlenet client. Each client downloads all of the data and reads what it can, using keys you have collected over time off of other users. Keeping it local means that no readable is stored on the server, so evil moderators cannot have their way with your data. Encrypted data is stored on the server, but nobody can read it who you didn't send it to.

4.1.5 How big does this database get?

As the only things being stored are text, not images or video, this means that each message is only small and will likely be less than a few megabytes over one year's very active use.

4.1.6 Why would someone want to build from source?

Given that compatibility of jars isn't an issue the only reason to do so is to ensure that your binaries derive from the public source code and not an evil secret version.

4.1.7 The Client does stuff I don't think it should do...

You may have found a bug for us accidentally. email to one of the addresses at the beginning of the user manual and the developers will have a look at it. As the source is being released, maybe the community will have a look and suggest a fix themselves.

4.1.8 What do Server Moderators of Turtlenet do?

We don't have any, we can't moderate content we can't see.

4.1.9 I want to mod Turtlenet. Can I have the source?

It's nice to know that others wish to take up the helm, pioneering a secure method of communication. You can have the source, it is available to the public to browse and modify.

4.1.10 Why choose 'X' over the clearly superior 'Y'?

As developers ourselves, we understand that other people have differing opinions. That's the joy of releasing code. Other people can pick up what we have done, or use our ideals as a starting point for their own thing. What this project stood for is ease of use for the end user and security from any unwanted external influences and this, we believe, is achieved.

Chapter 5

Side Notes

5.1 Java Database Connector

Our project uses the 'Java Database Connector' (JDBC) SQLite driver developed by 'Taro L. Saito'
- also known as xerial. Xerial's website is here: <http://xerial.org/>
if you wish to obtain the driver or otherwise contact and give support, then there is your person.

Part IV

Portfolio

Chapter 32

Deviations in Requirements and Design

1. We have decided that the event function isn't very valuable and so dropped it from the requirements early in development.
2. We decided that having a website and active servers was important and so added it early in development.
3. Our data flow has changed in that the client now updates the local database without waiting for updates from the server to arrive. This was done so that network latency didn't interfere with the user experience. All actions are still sent to oneself via the server, else multiple clients with the same key wouldn't function.
4. The client-client protocol has been significantly expanded so that all actions can be represented within it. This is so that if all one has is a keypair to an account, that account may be fully recovered. An example of new functionality in the protocol is that category creation and modification is recorded on the server (via encrypted messages sent, and viewable, solely to oneself). NB: The client-server protocol is wholly unaltered.
5. The datatypes in the database have changes due to the limitations of SQLite.
6. The primary key in many database tables has changed from an arbitrary value to a globally identifying cryptographic signature (from the message establishing the relevant datum.)
7. The database doesn't make use of foreign keys because the combination of network latency being potentially different for every message (due to Tor) and asymmetric relationships and communication means that foreign keys will often reference something that either doesn't exist

yet or will never exist. Furthermore in SQLite a foreign key can only reference one thing, and two of our potential foreign keys don't always reference the same field. Therefore there is only one possible foreign key in our entire database: tCategoryMembers.catID references tCategory.catID, and even that is tenuous at best as relies upon undefined behaviour of the frontend. For these reasons we removed foreign keys from the schema. The new database design is as follows:

8. A number of accessors were added to the Message class for extracting information from different types of messages.
9. The DBStrings class was added so as to keep SQL query strings separate from java code, and to localize them within one namespace. This class merely contains a large number of static strings.
10. The Logger, Tokenizer, MessageFactory, and F(file)IO class were added as helper classes. Tokenizer was created instead of using Java's existing Tokenizer class because we needed a tokenizer automatically convertible to javascript. A factory class was required because the Message class cannot contain constructors not automatically convertible to javascript.
11. The following data bearing classes were created to return structured data to the HTML/JS frontend. They allow the more powerful Java backend to extract (and format) data from the database before returning a simple class containing it.
 - (a) Friend
 - (b) CommentDetails
 - (c) Conversation
 - (d) PostDetails
12. The Turtlenet, TurtlenetImpl, and TurtlenetAsync classes exist to provide an asynchronous interface between the HTML/JS frontend and the Java backend.

32.1 Logical table design version 2.0

tCategory

catId	canSeePDATA
-------	-------------

tCategoryMembers

pk	catID	userKey
----	-------	---------

tClaim

sig	name	claimTime
-----	------	-----------

tRevocations

key	sig	timeOfLeak	creationTime
-----	-----	------------	--------------

tPostVisibleTo

pk	postSig	key
----	---------	-----

tConvoKeys

pk	convоИD	key
----	---------	-----

tConvos

convоИD	timeCreated
---------	-------------

tUser

key	username	knowName	email	name	gender	birthday
-----	----------	----------	-------	------	--------	----------

tComment

sig	msgText	senderKey	parent	creationTime
-----	---------	-----------	--------	--------------

tPost

sig	msgText	time	recieverKey	sendersKey
-----	---------	------	-------------	------------

tConvoMessages

pk	convоИD	sendersKey	msgText	time
----	---------	------------	---------	------

tEvent

sig	startTime	endTime	creatorKey	accepted	name	creationTime
-----	-----------	---------	------------	----------	------	--------------

tLike

pk	likerKey	parent
----	----------	--------

32.2 Database design version 2.0

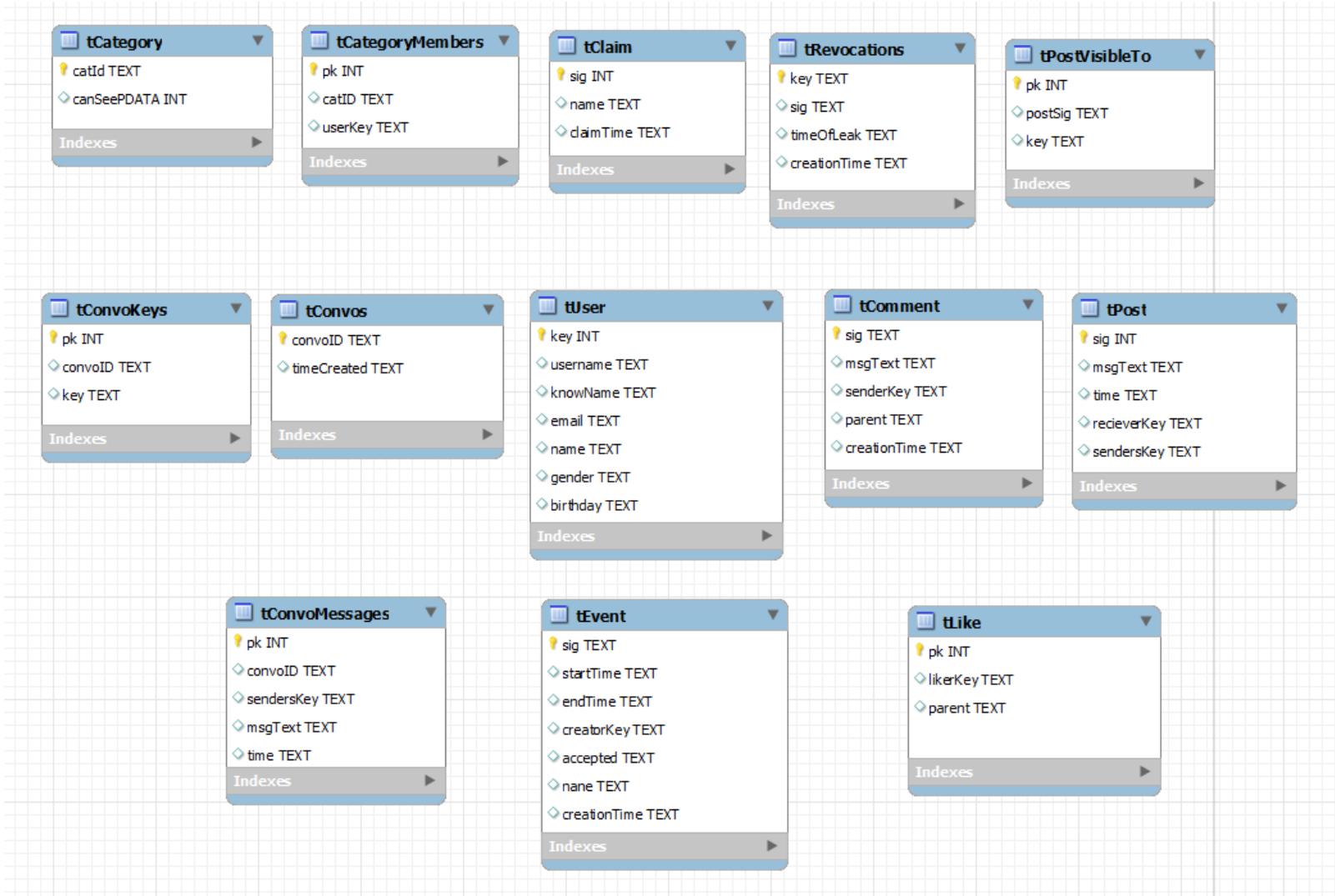


Figure 32.1: Database Entity Relationship diagram

32.3 Expanded Client-Client Protocol

Given number of new message types were added a summary of the new types can be found below:

Command	Format of content	Purpose
ADDCAT	true:family	add a new category named ‘family’ that can see your profile data
UPDATECAT	false:family	change the category named ‘family’ to be unable to see your profile data
ADDTOCAT	<key>:family	where ‘<key>’ is replaced with a UTF-8, base64, X509 encoded RSA public key it adds that key to the category named family
REMFROMCAT	<key>:family	where ‘<key>’ is replaced with a UTF-8, base64, X509 encoded RSA public key it removes that key from the category named family
ADDKEY	<key>	where ‘<key>’ is replaced with a UTF-8, base64, X509 encoded RSA public key it adds that key to your list of known people

Chapter 33

Turtlenet Website

The website (www.turtlenet.co.uk), allows users to download the client and view details on the project. For example, who has been working on it, and the fine print regarding the licensing. These things can not only be useful, but also necessary to inform the user what the program is, and why they need it, along with providing a suitable means of distribution for the project. Below is a short description of the website and its content.

- Home Page: This is a quick general description about what Turtlenet is, providing you with a link to both get the sourcecode for Turtlenet, and the client itself.
- Downloads: Holds the Turtlenet.jar file link, allowing the user to retrieve the client, as it does other pertinent information regarding any dependencies the client will need to run.
- About Us: This page is a small area dedicated to the team behind Turtlenet, Ballmer Peak, and our views on the project.
- Content: This area of the website contains all the contact information for our staff, should you need to contact them.
- FAQs: Frequently Asked Questions about the project come here.
- License: Finally, here is the legal rambling regarding the license for the project.

Chapter 34

Hosting

We chose Amazon Web Services(AWS) to host the remote server component of Turtlenet and the project website. The primary reason for this is that AWS offers a free package for the first 12 months of membership. The price after this is also fairly competitive although we would probably consider other providers if the offer didn't exist.

Another advantage of AWS is, due to their size, Amazon have many servers available which lends itself to high performance and increased redundancy. Performance isn't a major issue as the Turtlenet remote server acts as little more than a middleman. Still, the lower the latency the better.

It is uncertain if Amazon has provided any of it's customers personal data to the NSA or any other government sponsored agency. This isn't of concern though as very little data is stored by the Turtlenet remote server. Due to the distributed nature of Turtlenet all sensitive information is stored in a local database on the users device.

Chapter 35

Source Code and Github.com

We chose to store our source code in a Git repository hosted by Github at: <https://github.com/thomal/COMP208/>.

The repository records all changes to files which allows group members know who has modified files, what has been added and what has been deleted.

If more than one person edits the same file it is possible to merge together the work of each person, often automatically. When more than one person edits the same line manual intervention is required to see who's edit should be kept.

We can also look back at previous revisions of files to figure out where bugs were introduced or to resurrect old code that has since been removed but may still be useful.

Chapter 36

Testing

36.1 Automated Testing

We have included a series of automated tests with Turtlenet. These tests are ran automatically whenever the binaries are built, they can be ran manually with ‘make test’.

For example we can test the part of the system responsible for parsing the contents of wall posts by constructing a wall post Message instance and calling POSTgetKeys(). If we do not receive the expected value for any test the developer is alerted.

More examples of the automated testing we used can be found at: <https://github.com/thomal/COMP208/blob/master/src/ballmerpeak/turtlenet/testing/Test.java>.

36.2 Black-box Testing

Black-box testing comprised mainly of:

- Clicking on buttons even if they didn’t fit into the logical work flow.
- Attempting to ‘cheat’ the system by making it do things that aren’t intended by the design.
- Entering silly values into various parts of the system in an attempt to see if this would break anything.

The system functioned as expected in all cases. Our intent in not validating the date field was because people don’t read things telling them how to format dates but rather type them how they normally do; however it allows for peoples date of birth to be “the past”. This perhaps isn’t such a bad thing, but was ultimately unintentional.

36.3 Usability Testing

We had a person who had never used Turtlenet before use the system with only five minutes of introduction beforehand. Without further instruction she was able to easily register, add my public key that I sent her, and take part in a conversaiton. She did however need help finding the wall of a friend. After I showed her she found it easy to do so after.

From this we've concluded that it's preferable to have more obvious ways of accessing anothers wall. This test occurred too late in development to add the change.

Chapter 37

Future Development

37.1 Interface Framework

Currently the interface framework is the Google Web Toolkit (GWT). This currently allows the Turtlenet client's interface to run in a web browser of the user's choosing. This choice helped the group and the project initially by allowing the developers to not consciously worry about any particular problems in the running of the project in terms of the front end - our mentality was that everyone had a browser on their computers so it was one less requirement to add to the minimum requirements section of the user manual. In hindsight however, what this decision has done is open the project to a flood of potential bug reports; some from inexperienced users hoping for a fix on Internet Explorer whilst others filling in reports about how their personally compiled browser is not rendering a button, for example. With Java, the code is run by a Java Virtual Machine (JVM) so there is only one platform to worry about whereas through the use of a browser being a container for the user interface, each with different layout engines and capabilities, the project can expect to receive bug reports from at least four different front ends.

With the potential workload heavily increased just by a decision made by the user, which we assume holds little to no knowledge and just wishes the project work on their computer, this may be a problem for the future developers of Turtlenet. As a possible solution, a different framework could be used which provides an executable that is tailored to the operating system, as opposed to a web browser. Running in a native window as opposed to a browser means that there is only a couple of different ways the front end can appear, variations mainly appearing due to a change of operating system - although a different desktop environment (DE) would also affect the final look of Turtlenet. This improves the usability of the project as the final outcome would be an executable file, which runs in the same or similar manner to other programs on their computer, improves consistency as GWT creates JavaScript as a core component in the running of the front-end, which may be blocked

by the browser as well as creating specific statements which are interpreted or ignored dependent on the browser currently in use. Finally there is debugging, which is improved because there are fewer main variables for the testers to cater to, such as versions of browsers and their plug-in files which may hamper use of the front end. Through consolidation into a native window, testing and bug catching can become more efficient as there are less programs to load, and less time required.

37.2 Interface and House Style

The current interface for the client has been made in such a way that it provides all of the functionality of the project in fairly easily identifiable sections. What it does not do though is look polished enough to be on an average user's computer yet. This is most likely due to time constraints stressing for functionality as opposed to aesthetics. Another potential problem is the house style of the front-end. Green has symbolic meaning and was not chosen simply due to the name of the project - Turtles more often than not have darker colours such as brown or grey and not green. Green is often used in healthcare as a sign that something is either safe or good for you (the green health 'plus' being an example), which is what Turtlenet aims to be for your communicative efforts. On a per-user basis however, colour is simply a way of making the front end become more pleasant.

This leads to the problem of personal taste - some people don't like green. Therefore to increase usability of the project as well as the total amount of users, themes could be a future development. Allowing the user to change the look of the front end can make a difference to the amount of people using the project. More users may appear if the project synchronises well with the rest of their system.

37.3 Languages Used

The project used Java for the back end of the system, SQLite for the Database and Java converted to JavaScript for the front end. Java was chosen for the interoperability of the language - being able to run on whatever has a Java Virtual Machine (JVM), which are available for most operating systems. Most users have the Java Runtime Environment (JRE) installed, which includes JVM so Java was a good choice for the project.

SQLite is a notably lightweight Database Management System (DBMS) at the expense of some features that are used in a more complete SQL solution, none of which were needed for the project. SQL notably requires you to define data type as well as the length of the variable as well - while SQLite is more lenient in this regard, removing this constraint completely would make a system more usable. An example of a more user friendly database would be one that uses MongoDB but that is not as popular as an SQL derived DBMS, so this is the reason SQLite was chosen.

Google Web Toolkit (GWT) allowed one of our developers the capability of writing code in a similar manner to Java which when compiled creates the required JavaScript and Ajax code. On a technical level we believed this to be quite clever, and was one of the reasons we chose GWT for the interface framework. In hindsight it would have been better to choose a different framework which would allow us to get a native executable after compilation. This would mean that the user does not need to open a terminal and enter any Java commands, improving usability by not forcing the user to enter an environment that they are not comfortable with.

Chapter 38

Ballmer Peak

Ballmer Peak is the team behind Turtlenet, both its designers and creators. We have worked together over the past 6 months to create a new, innovative product the entire team is proud of. Pooling our resources and individual technical skills we feel we came together as a team and produced a quality product. We all had our roles to fill in this project:

38.1 Peter Duff - Team Lead

- In charge of project management and organisation, as well as assigning roles and responsibilities, amongst other generic managerial admin.
- Was the designer and creator of the Turtlenet website.
- Proofreader of the entire requirements, design and portfolio documents.

38.2 Luke Thomas - Lead Programmer

- Designed and created the initial prototype and idea for the project.
- Wrote the whole of the client, except for Database.java, DBStrings.java, and frontend.java. Significantly contributed to Database.java and frontend.java.
- Wrote the whole of the server.
- Wrote the automated tests, designed L^AT_EX documents.

38.3 Louis Prince - Programmer

- Programmed and designed the GUI for the client (frontend.java).
- Redesigned parts of the Turtlenet website.

38.4 Aishah Senin - Database Developer

- Documented all aspects of the database.
- Designed and created the SQLite database from scratch.
- Wrote database queries.

38.5 Michael Chadwick - Software Engineer

- Wrote the majority of the user manual for the program.
- Provided heavy documentation regarding some of the finer, non-technical points of the project (Gantt, Risk Asses. etc.).
- Wrote a significant portion of Database.java

38.6 Leon Choi - Researcher

- Main researcher into existing similar systems and potential future work.
- Designed a mock GUI for future android work.

Chapter 39

Professional Issues

1. Lack of communication Communication is highly important when working as a team. It allows team members to be aware on what is happening as a whole on the project at hand. Despite working on one area of a project, there are always some areas where there is a need to integrate the work to other areas, and this is when communication comes in handy. Team members are also expected to keep each other updated frequently regarding the progress of the project so no one is left behind. Other reasons why communication is vital is ideas can come from different members, making sure that these ideas are reached out and heard and possibly put into account brings team members to feel appreciated and important.
2. Being open towards one another There will always be a fallout between teams over time. To avoid that, team members are advised to be open towards others especially when there is a problem on certain areas. Criticism is encouraged, but as are good manners.
3. Handling of non-team players What can the other team members do when someone in the team exhibits such behaviour? One way is to encourage the team member to modify his or her behaviour for the sake of others. Inform the person regarding the reason why we are working as a team, and changing such behaviour might benefit the team as a whole. Team members who displays such behaviour may create negative feelings from other team members which is not healthy for the relationship of the team.

Chapter 40

Achievement of Objectives

We successfully produced a usable social network requiring minimal knowledge of cryptography that can be used to hide both what one says and whom one says it to. We did not however allow users to transfer public keys as QR codes, rather we made them copy/paste a string of text. As an extra feature we added Tor support to hide the originating IP addresses of traffic, while this isn't as secure as our other measures it is secure in practice, with the NSA being unable to deanonymise Tor users[30]. Our pages do not auto-refresh at the moment, meaning that to see new content a user must reload the page manually. On the other hand we added the feature of encrypting all information stored locally, and extended the protocol so that with only a users keypair their entire local database may be rederived on any device; this is useful for users with multiple devices.

On the whole the missed some objectives and added a good deal more. We feel that what we gained was far greater than what we lost, and so was worth the time we spent on it in lieu of the unimplemented, but planned, features.

Chapter 41

Evaluation

I think the best evaluation is that we didn't expect this. This was a thoroughly challenging project, that we did not anticipate being so large when we started. Typically as a student, we found university modules to have a similar workload to each other through the university courses so far. This group project however was a different thing entirely. The work hours put in for this module eclipsed all other modules by far, even combined in some parts. This being said, it has also been rewarding to many of us, to work together producing a much bigger result than the small projects we normally create. Self learning was also widely needed during this module in order to teach us all the skills required to complete the project.

Overall though, we feel the project has been a great success. With a live website running right now (www.turtlenet.co.uk), live working repository (<https://github.com/thomal/COMP208>), a product that works fully functioning as intended, its hard not for us to be satisfied. Even despite not meeting all our original requirements, we produced a functional social network that aptly serves to complete our original target goal of a secure social network, that holds privacy in its highest regard. To say things went perfectly however, would be a fallacy.

There are areas we could have definitely improved upon, such as the general teamwork between members. The idea of group study sessions unfortunately came too late in the project to be done, but the idea of us all taking a weekend to work on the project together on one desk would have likely yielded positive results. Better work deadlines would have also been useful, more frequent, smaller workloads may have well served better than larger apart ones. We should have also done more research on what to produce the client UI with, as it turns out GWT was not the best choice. Apparently it is quite limiting to use and better alternatives would have been a more optimal choice.

We did not make all bad decisions though, the idea of abandoning the provided university workspace in favour of git was an astounding success, and we are all convinced reproducing the same project in that workspace would simply not be possible due to the complexity of the program.

www.github.com provided us with the means to make the project possible, and documents our project from the start step by step, an excellent tool for group projects such as this. We are all in agreement version control is a must for group projects of this scale.

In summary, the project was successful, and something for us as a team to be proud of. It is something many of us wish to continue working on even after this module is all over. It is something that taught us more skills than any other module. And most of all it is something for all to have, a gift to the world pool of knowledge, accessible to all at no cost, and perhaps one step forward in a fair, free internet.

Appendices

Appendix A

Minutes

A.1 Meeting #1 Minutes (Thursday, 30/01/2014)

Peter, Luke, Aishah, Leon, Mike

- Introductions
- Overview of the project
- Assigned roles to members

A.2 Meeting #2 Minutes (Friday, 31/01/2014)

Peter, Luke, Aishah, Leon, Mike

- We ate nice chinese in celebration of the new year
- If I'm honest this wasnt really a team meeting, more a hunger thing

A.3 Meeting #3 Minutes (Tuesday, 04/02/2014)

Peter, Luke, Aishah, Leon, Mike

- State out the problems, criticisms on Facebook regarding user privacy issues (Leon)
- Data flow of the system (Luke)
- User requirements (Luke has done the draft. Refinement to be done by Aishah and Peter)

- Class diagram (to be completed after dataflow diagram and user requirements)
- Sketches of GUI (Peter and Mike to do this together)
- GANTT chart and risk assessment (after user requirements has been drafted out)
- Data dictionary (Aishah)
- Read about how to implement SQLite (Aishah)

A.4 Meeting #4 Minutes (Friday, 07/02/2014)

Peter, Luke, Aishah, Leon, Louis

- Introduced Louis Prince to members/project
- Assigned Roles to requirement sections
- Team Review date proposed (Wed 19th, Afternoon)
- Team name: Ballmer Peak

A.5 Meeting #5 Minutes (Wednesday, 12/02/2014)

Peter, Luke, Aishah, Leon, Mike

- Allocated left over parts
- Feedback on project and requirements so far
- Project name: Turtlenet
- Louis Prince absent from scheduled meeting.

A.6 Meeting #6 Minutes (Wednesday, 19/02/2014)

Peter, Luke, Aishah, Leon, Mike, Louis

- Discussed design phase, outlined what needs to be done
- Also outlined who needs to do it, tasklist:
- Mike - Use Case Diagram, Data Dictionary
- Leon - Mobile GUI, Sequence Diagram

- Louis - Web GUI Design, Java/SQLite/HTML-CSS Documentation
- Aishah - Database Design Doc
- Peter - Swing/AWT GUI Design, Server GUI Design
- Luke - Class Interfaces, Protocol, Architecture, Data Flow Diagrams, More Protocol, Psuedocode

A.7 Meeting #7 Minutes (Friday, 07/03/2014)

Peter, Luke, Aishah, Leon, Mike, Louis

- Business Rules (Peter)
- Gantt Chart (Mike)
- Various DB tweaks (Aishah)
- Merge work into PDF (Luke)
- Rename GUI design as storyboard

A.8 Meeting #8 Minutes (Wednesday, 19/03/2014)

Peter, Luke, Aishah, Leon, Louis

- Server.java (Luke)
- Client.java (Luke)
- Crypto.java (Luke)
- NetworkConnection.java (Luke)
- Parser.java (Luke)
- HTTPSserver.java (Luke)
- helper classes (Luke)
- browser plugins (Luke)
- QR Code parser (Luke)
- Test harness (Luke)

- Installer (Peter)
- Website (Peter [mimic gnome.org])
- Manual (Peter)
- Hardware Server (Peter)
- ServerGUI.java (Leon)
- First run config (Leon)
- Database.java (Mike, Aishah)
- SQLite Database (Aishah)
 - Database connection
 - Create DB
- Logo and Graphic Design (Aishah)
- GWT interface (Louis)
 - Stubs in interface
 - error on failure to connect
 - Add public keys
 - Categorise users
 - Post to your wall
 - Read others wall posts
 - Post to anothers wall
 - Events create and receive
 - Chat
 - Comment posts and comments
 - Like posts and comments

A.9 Meeting #9 Minutes (Friday, 21/03/2014)

Peter, Luke, Aishah, Mike, Louis

- Everyone can build project
- DB Connection
- GWT Interface stubs
- Stubs for other classes
- Manual contents page
- Create table statements
- Compile remote server as a JAR
- Server GUI, start up and shut down

Leon absent.

A.10 Meeting #10 Minutes (Friday, 28/03/2014)

Peter, Luke, Aishah, Leon, Mike, Louis

- Fix manual stuff, add makefile, finish initial test setup (luke)
- Create tables from Database.java (aishah/mike)
- GWT Interface (louis)
- Start/Stop server via gui (leon)
- Webstie Prototype (peter)
- Document regarding hashing for various classes (luke)

A.11 Meeting #11 Minutes (Sunday, 13/04/2014)

Peter, Luke, Aishah, Louis

- Progress Recap: (Luke)
 - Added testing

- Added logger
- Implemented createDatabase to execute aishahs create table SQL queries
- Message::XgetY methods
- Altered frontend to start turtlenet when it starts, and stop it when the tab is closed
- Re: Enable -strict for compiling frontend (Luke/Louis)
- Installer (Luke)
- Store the signature (String) for posts and comments (Aishah/Mike)
- SQL for database methods (Aishah/Mike)
- Implement database methods (Aishah/Mike)
- Start/Stop server via gui (Leon)
- Website Prototype (Peter)
- Call appropriate Database.getX methods (Louis)

Leon, Mike not present.

A.12 Meeting #12 Minutes (Saturday, 19/04/2014)

Peter, Luke, Aishah, Leon, Mike, Louis

- Website finally unveiled
- Reassignment of some tasks (prioritization)
- Mostly a formality to remind people of deadlines

A.13 Meeting #13 Minutes (Friday 02/05/2014)

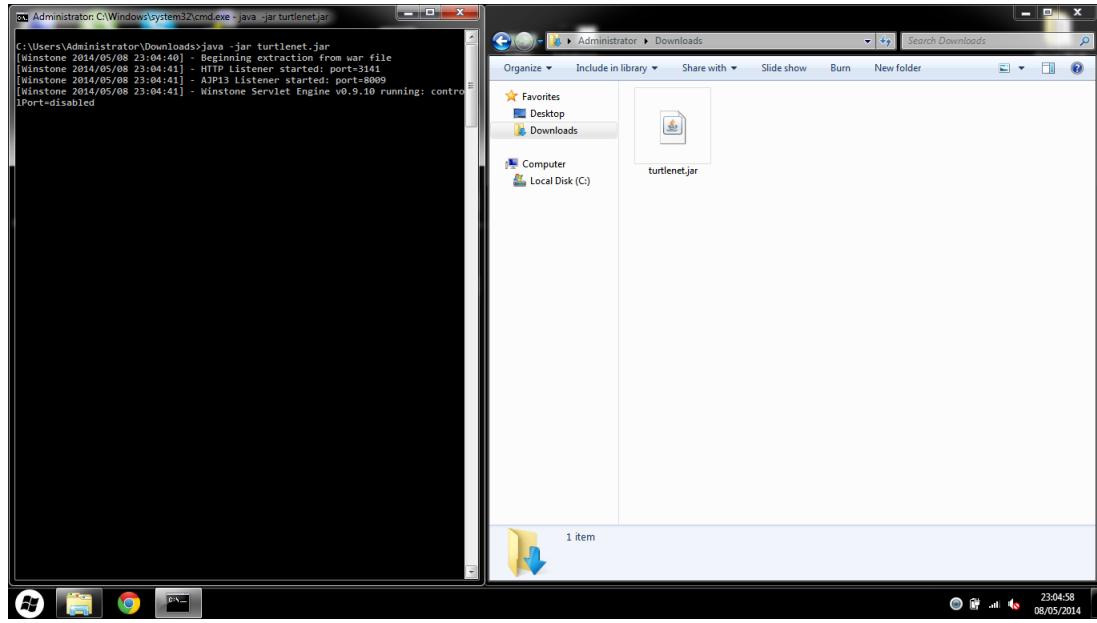
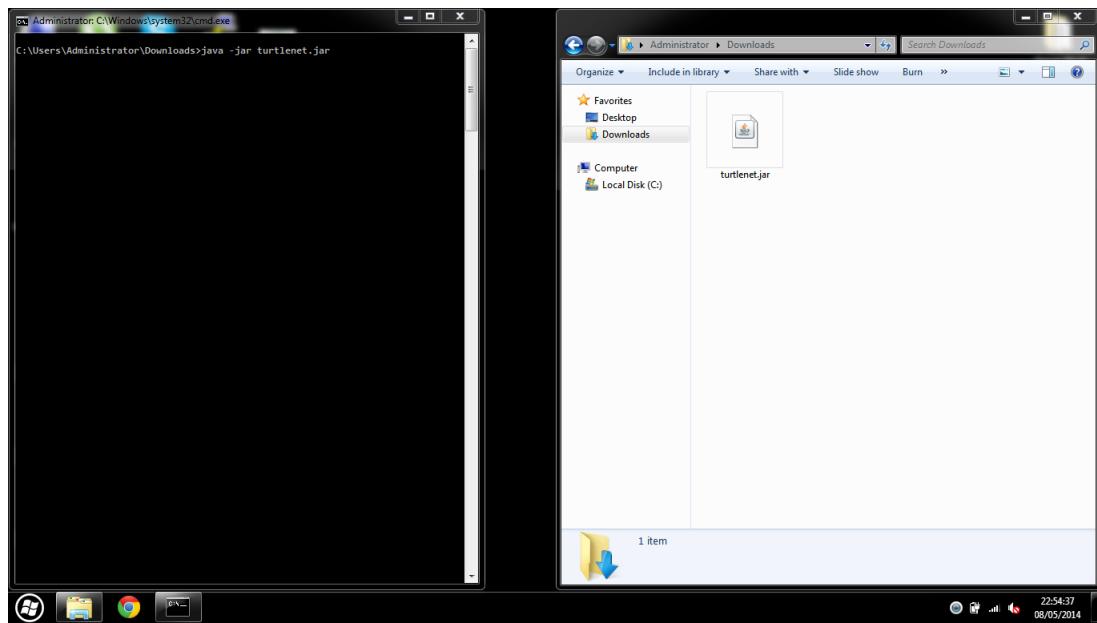
Peter, Luke, Aishah, Leon, Mike, Louis

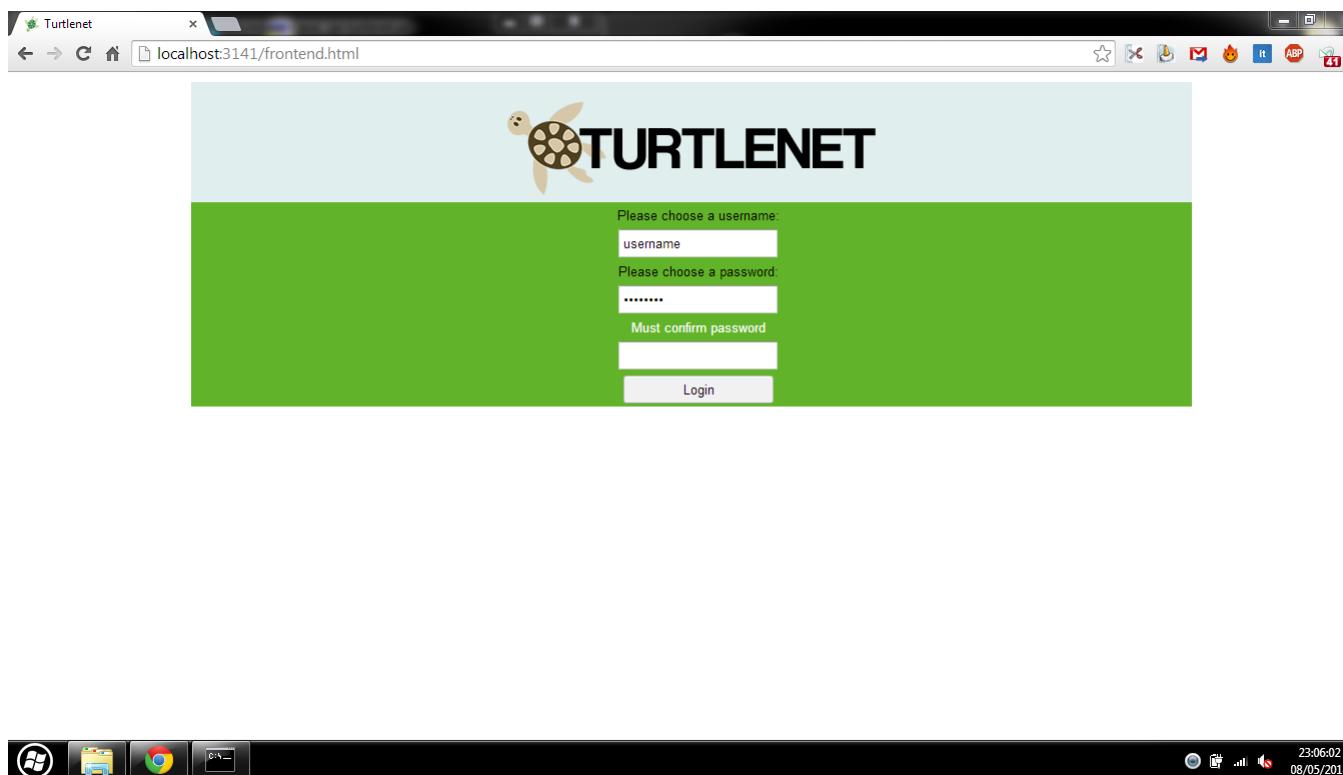
- Assigned final weeks tasks and roles for submitting the portfolio
- Updates Requirements (Aishah)
- Updates Design (Aishah)
- Website (Leon)

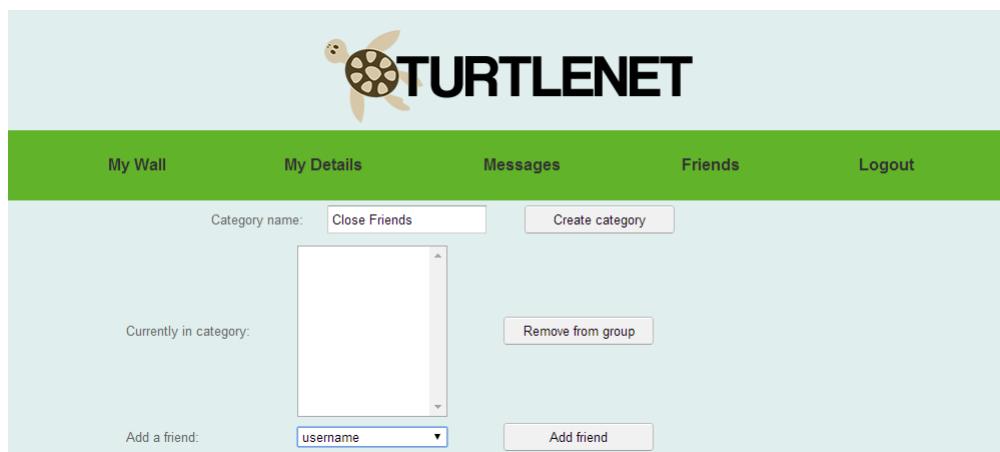
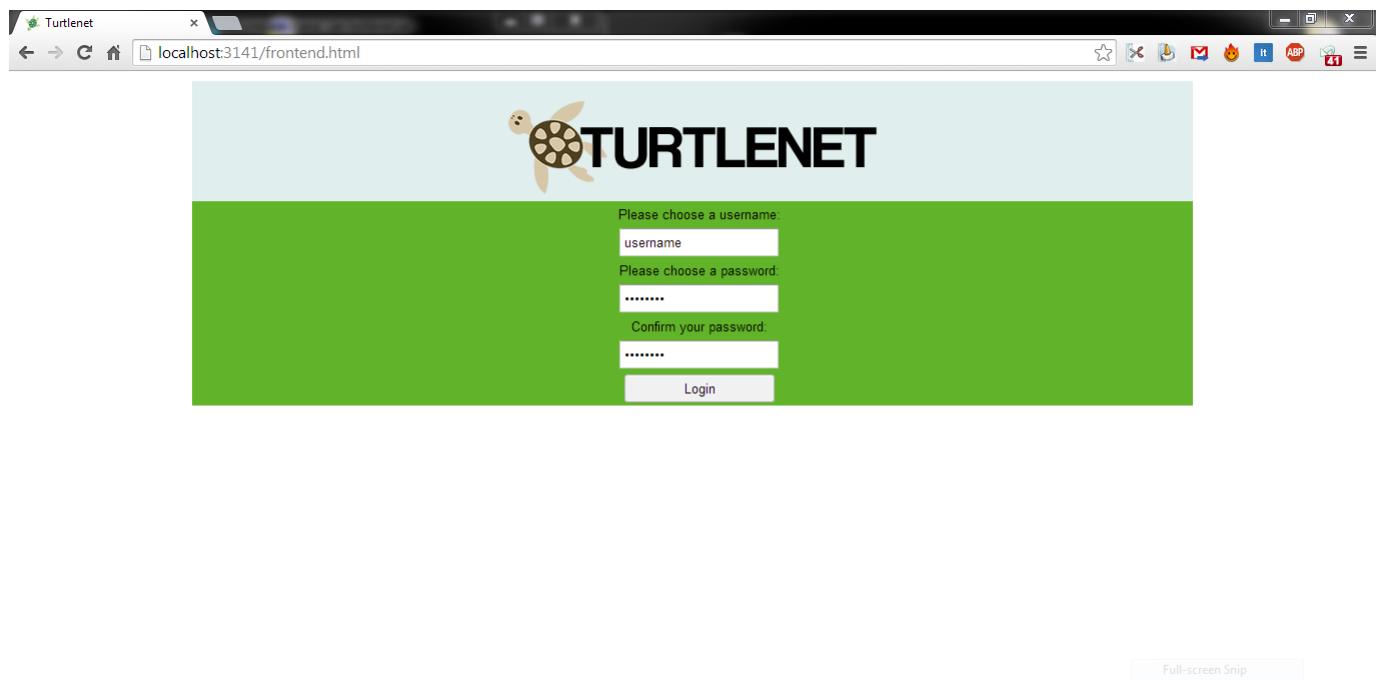
- Source Code (Leon)
- Personal Statements (All)
- Deviations Requirements (Luke)
- Deviations Design (Luke)
- Hashes (Luke)
- User Manual (Mike)
- Future Development Continued (Mike)
- Hosting AWS (Louis)
- Testing Automated/Blackbox (Louis)

Appendix B

Screenshots







The screenshot shows the TURTLENET messaging interface. At the top, there is a logo of a sea turtle and the word "TURTLENET". Below the logo is a green navigation bar with links for "My Wall", "My Details", "Messages", "Friends", and "Logout". The main area has a light blue background. On the left, there is a vertical white box labeled "Username". In the center, a message box contains the text "Hello, this is a message". To the right of the message box is a "Send" button. At the bottom, there is a "Choose a friend:" dropdown menu with "username" selected, and a "Add to the conversation" button.

The screenshot shows the TURTLENET wall posting interface. At the top, there is a logo of a sea turtle and the word "TURTLENET". Below the logo is a green navigation bar with links for "My Wall", "My Details", "Messages", "Friends", and "Logout". The main area has a light blue background. There are two main sections. The top section is a post box with a blue border, containing the text "This is another post!". It includes buttons for "About Me" and "Updates paused". Below this is a "Send" button. The bottom section is a comment box with a pink border, containing the text "This is a post". It includes a "Cancel" button and a "Send" button. At the bottom of the page, there are buttons for "Add a comment" and "Like".



TURTLENET

[My Wall](#) [My Details](#) [Messages](#) [Friends](#) [Logout](#)

[Reload page](#) [Write a post](#)

Username:	username
Name:	User Name
Birthday:	01/01/1970
Gender:	Female
Email:	example@example.com
Public Key:	MIGMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDizJ7GzWJ/3I

[Edit my details](#)

This is another post

Page auto update paused [Unlike](#)

This is a comment

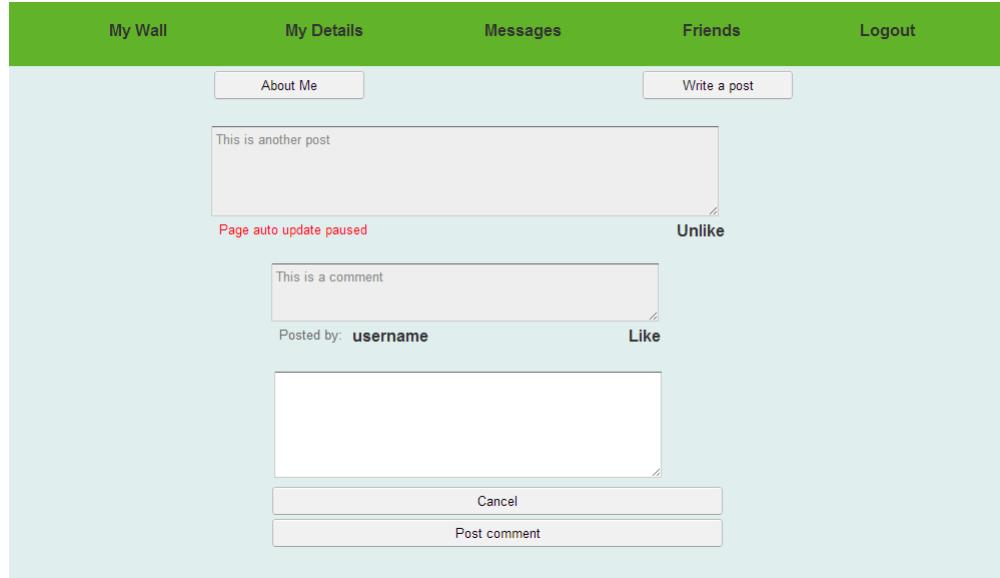
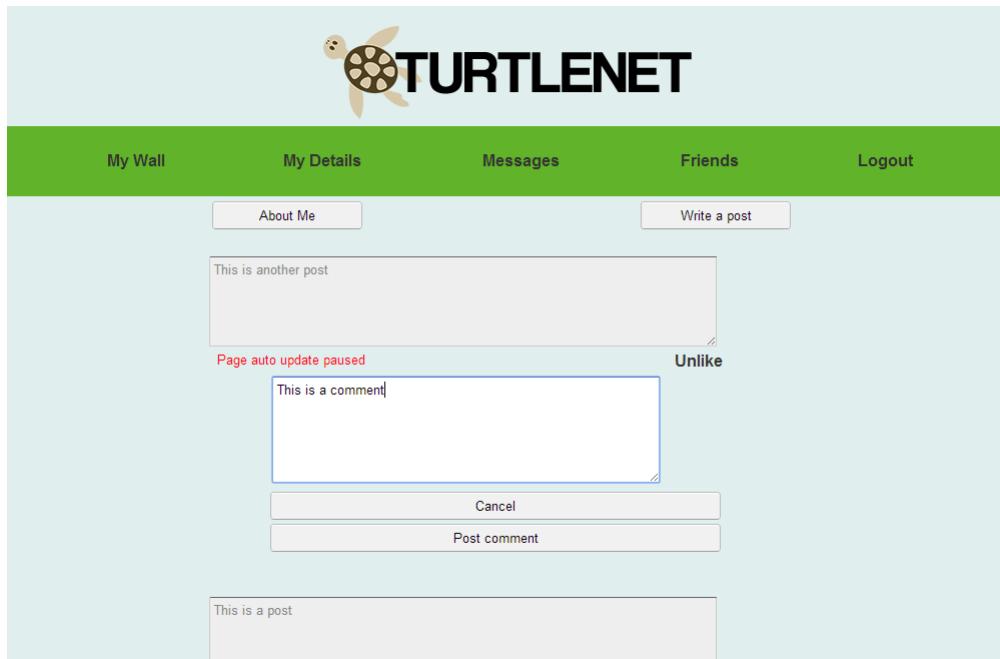
Posted by: **username** [Like](#)



TURTLENET

[My Wall](#) [My Details](#) [Messages](#) [Friends](#) [Logout](#)

Category name: [Create category](#)





TURTLENET

My Wall My Details Messages Friends Logout

About Me Write a post

This is another post

Add a comment Unlike

This is a post

Add a comment Like



TURTLENET

My Wall My Details Messages Friends Logout

Friend's Name Friend's Public Key

username Add new friend
Add new category

My key: All ▾



TURTLENET

My Wall My Details Messages Friends Logout

Username:	<input type="text" value="username"/>	<input type="button" value="Save Username"/>	
Name:	<input type="text" value="User Name"/>	<input type="button" value="Save Name"/>	Name saved
Birthday:	<input type="text" value="01/01/1970"/>	<input type="button" value="Save Birthday"/>	Birthday saved
Gender:	<input type="text" value="Female"/>	<input type="button" value="Save Gender"/>	Gender saved
Email:	<input type="text" value="example@example.com"/>	<input type="button" value="Save Email"/>	Email saved
<input type="button" value="Revoke Key"/>			

If you revoke your key your account will be deleted!
Select which groups can view your details:



TURTLENET

My Wall My Details Messages Friends Logout

Hello, this is a message
A second message

username
username

New conversation



TURTLENET

My Wall My Details Messages Friends Logout

Participants:

username A second message
username Really? A second message?
username Yes, a second message!

Reply



TURTLENET

My Wall My Details Messages Friends Logout

Enter the key of the person you wish to add:

MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDizJ7GzWJ3hVwSVDrI0Qe5SGR3vb0rXQ2RDKKaS7

Add key

Appendix C

Deadlines

- **2014-01-31** topic and team
- **2014-02-14** requirements
- **2014-03-14** design
- **2014-05-09** portfolio & individual submission

Appendix D

TODO

D.1 General

Auto-Refresh the page on new content.

D.2 Requirements Weeks 1-3

1. Project Desc.
 - **COMPLETE** Project being done for (Peter)
 - **COMPLETE** Mission Statement (Luke)
 - **COMPLETE** Mission Objective (Luke)
 - **COMPLETE** Threat Model (Luke)
2. Statement of Deliverables
 - **COMPLETE** Desc. of anticipated documentation (Luke)
 - **COMPLETE** Desc. of anticipated software (Aishah)
 - **COMPLETE** Desc. + Eval. of any anticipated experiments + blackbox (Louis)
 - **COMPLETE** User view and requirements (Luke)
 - **COMPLETE** System requirements (Luke)
 - **COMPLETE** Transaction requirements (Aishah)

3. Project and Plan

- **COMPLETE** Facebook research (Leon)
- **COMPLETE** Case Study: Tor (Luke)
- **COMPLETE** Case Study: alt.anonymous.messages and mix networks (Luke)
- **COMPLETE** Case Study: PGP and E-Mail (Luke)
- **COMPLETE** Implementation Stage (Peter)
- **COMPLETE** Milestone Identification (Milestones can most easily be recognised as deliverables) (Mike)
- **COMPLETE** Gantt Chart (Mike)
- **COMPLETE** Risk Assessment (Mike)

4. Bibliography

- **COMPLETE** Bibliography framework (Luke)
- **COMPLETE** Add citations where relevant (Everyone, in their own sections)

D.3 Design Weeks 4-X

- **COMPLETE** Use Case Diagram (Mike)
- **COMPLETE** Glossary (Mike)
- **COMPLETE** Mobile GUI Design (Leon)
- **COMPLETE** Sequence Diagram (Luke/Leon)
- **COMPLETE** HTML GUI Design (Louis)
- **COMPLETE** DB Design (Aishah)
- **COMPLETE** Transaction Design (Aishah)
- **COMPLETE** Server GUI Design (Peter)
- **COMPLETE** Class Interfaces (Luke)
- **COMPLETE** Protocol (Luke)

- **COMPLETE** Architecture (Luke)
 - **COMPLETE** Data Flow Diagrams (Luke)
 - **COMPLETE** Pseudocode (Luke)
 - **COMPLETE** Class Diagram (Luke)
5. Portfolio
- **COMPLETE** Deviations from requirements and design (Luke)
 - **COMPLETE** Sample screenshots (Luke)
 - **COMPLETE** Test strategy and results (Louis)
 - **COMPLETE** Description of team members and roles (Peter)
 - **COMPLETE** Achievement of objectives (Luke)
 - **COMPLETE** Evaluation (Peter)
 - **COMPLETE** Professional Issues (Peter)
6. Other
- **COMPLETE** Licensing (Luke)
 - **COMPLETE** Source Listing (Luke)
 - **COMPLETE** Minutes (Peter)
 - **COMPLETE** Individual Submission (Luke)
 - **COMPLETE** Individual Submission (Peter)
 - **COMPLETE** Individual Submission (Mike)
 - **COMPLETE** Individual Submission (Louis)
 - **COMPLETE** Individual Submission (Aishah)
 - **COMPLETE** Individual Submission (Leon)
 - **INCOMPLETE** Programmers Reference/Documentation (Luke)

Appendix E

Bugs Accidental Features

- Entering the wrong password erases the database.

Appendix F

Licence

Ballmer Peak (i.e.: The students which worked on Turtlenet) have waived their rights via the CC0 licence. The university of liverpool now has sole claim to limiting rights placed upon Turtlenet (NB: Not sole rights to it, the effect of the CC0 waiver is to grant all the rights we had to everyone. The limitations placed upon Turtlenet by The University Of Liverpool now apply to any user of Turtlenet to the same extent they did to us before the waiver.).



To the extent possible under law, Ballmer Peak has waived all copyright and related or neighboring rights to Turtlenet and Associated Documentation. This work is published from:
United Kingdom.

F.1 Statement of Purpose

The laws of most jurisdictions throughout the world automatically confer exclusive Copyright and Related Rights (defined below) upon the creator and subsequent owner(s) (each and all, an "owner") of an original work of authorship and/or a database (each, a "Work").

Certain owners wish to permanently relinquish those rights to a Work for the purpose of contributing to a commons of creative, cultural and scientific works ("Commons") that the public can reliably and without fear of later claims of infringement build upon, modify, incorporate in other works, reuse and redistribute as freely as possible in any form whatsoever and for any purposes, including without limitation commercial purposes. These owners may contribute to the Commons to promote the ideal of a free culture and the further production of creative, cultural and scientific

works, or to gain reputation or greater distribution for their Work in part through the use and efforts of others.

For these and/or other purposes and motivations, and without any expectation of additional consideration or compensation, the person associating CC0 with a Work (the "Affirmer"), to the extent that he or she is an owner of Copyright and Related Rights in the Work, voluntarily elects to apply CC0 to the Work and publicly distribute the Work under its terms, with knowledge of his or her Copyright and Related Rights in the Work and the meaning and intended legal effect of CC0 on those rights.

F.2 Copyright and Related Rights

A Work made available under CC0 may be protected by copyright and related or neighboring rights ("Copyright and Related Rights"). Copyright and Related Rights include, but are not limited to, the following:

1. the right to reproduce, adapt, distribute, perform, display, communicate, and translate a Work;
2. moral rights retained by the original author(s) and/or performer(s);
3. publicity and privacy rights pertaining to a person's image or likeness depicted in a Work;
4. rights protecting against unfair competition in regards to a Work, subject to the limitations in paragraph 4(a), below;
5. rights protecting the extraction, dissemination, use and reuse of data in a Work;
6. database rights (such as those arising under Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, and under any national implementation thereof, including any amended or successor version of such directive); and
7. other similar, equivalent or corresponding rights throughout the world based on applicable law or treaty, and any national implementations thereof.

F.3 Waiver

To the greatest extent permitted by, but not in contravention of, applicable law, Affirmer hereby overtly, fully, permanently, irrevocably and unconditionally waives, abandons, and surrenders all of Affirmer's Copyright and Related Rights and associated claims and causes of action, whether now

known or unknown (including existing as well as future claims and causes of action), in the Work (i) in all territories worldwide, (ii) for the maximum duration provided by applicable law or treaty (including future time extensions), (iii) in any current or future medium and for any number of copies, and (iv) for any purpose whatsoever, including without limitation commercial, advertising or promotional purposes (the "Waiver"). Affirmer makes the Waiver for the benefit of each member of the public at large and to the detriment of Affirmer's heirs and successors, fully intending that such Waiver shall not be subject to revocation, rescission, cancellation, termination, or any other legal or equitable action to disrupt the quiet enjoyment of the Work by the public as contemplated by Affirmer's express Statement of Purpose.

F.4 Public License Fallback

Should any part of the Waiver for any reason be judged legally invalid or ineffective under applicable law, then the Waiver shall be preserved to the maximum extent permitted taking into account Affirmer's express Statement of Purpose. In addition, to the extent the Waiver is so judged Affirmer hereby grants to each affected person a royalty-free, non transferable, non sublicensable, non exclusive, irrevocable and unconditional license to exercise Affirmer's Copyright and Related Rights in the Work (i) in all territories worldwide, (ii) for the maximum duration provided by applicable law or treaty (including future time extensions), (iii) in any current or future medium and for any number of copies, and (iv) for any purpose whatsoever, including without limitation commercial, advertising or promotional purposes (the "License"). The License shall be deemed effective as of the date CC0 was applied by Affirmer to the Work. Should any part of the License for any reason be judged legally invalid or ineffective under applicable law, such partial invalidity or ineffectiveness shall not invalidate the remainder of the License, and in such case Affirmer hereby affirms that he or she will not (i) exercise any of his or her remaining Copyright and Related Rights in the Work or (ii) assert any associated claims and causes of action with respect to the Work, in either case contrary to Affirmer's express Statement of Purpose.

F.5 Limitations and Disclaimers

1. No trademark or patent rights held by Affirmer are waived, abandoned, surrendered, licensed or otherwise affected by this document.
2. Affirmer offers the Work as-is and makes no representations or warranties of any kind concerning the Work, express, implied, statutory or otherwise, including without limitation warranties of title, merchantability, fitness for a particular purpose, non infringement, or the absence of

*F.6. LIMITATIONS PLACES UPON TURTLENET BY THE UNIVERSITY OF LIVERPOOL*185

latent or other defects, accuracy, or the present or absence of errors, whether or not discoverable, all to the greatest extent permissible under applicable law.

3. Affirmer disclaims responsibility for clearing rights of other persons that may apply to the Work or any use thereof, including without limitation any person's Copyright and Related Rights in the Work. Further, Affirmer disclaims responsibility for obtaining any necessary consents, permissions or other rights required for any use of the Work.
4. Affirmer understands and acknowledges that Creative Commons is not a party to this document and has no duty or obligation with respect to this CC0 or use of the Work.

F.6 Limitations places upon Turtlenet by The University of Liverpool



Policy on Exploitation and Commercialisation of Intellectual Property

January 2012

The Policy

1. INTRODUCTION

1.1 Intellectual Property Rights (IPR) are the legal rights associated with creative and intellectual effort or commercial reputation and goodwill. Together, the different types of intellectual property rights (patents, copyright, know-how, registered trademarks, registered designs and unregistered designs) protect a wide variety of property including literary and artistic works, computer programs, inventions, designs and marks used to identify goods and services. Intellectual property (IP) can be a valuable asset and substantial income can be generated through its successful exploitation. The nature of the University's activities, in particular its research activities, often gives rise to the creation of valuable IP which can benefit the University, its staff and students, as well as third parties.

2. OWNERSHIP OF IP

2.1 The following section clarifies the University's position with regard to IP ownership.

Employees

2.2 Under English law, notably the Patents Act 1977 and the Copyright, Designs & Patents Act 1988, IP generated in the course of a person's normal employment belongs to the substantive employing organisation. Thus, in the absence of any agreements to the contrary, the University assumes ownership of all IP generated by staff.

2.3 The University, however, does not usually intend to assert ownership of copyright in books, articles, lectures and artistic works, other than those which are capable of commercial exploitation, specifically commissioned by the University or are of benefit to a partner institution of the University, for example Xi'an Jiaotong-Liverpool University (XJTLU). In such cases, Level 1

Heads will decide the use of such works and, where required, will be offered training to help them exercise that judgement.

- 2.4 The University does not, however, relinquish ownership of copyright in computer software.
- 2.5 It is generally accepted that most academic staff wish to publish articles in learned journals. As a general rule the University will preserve the rights of academic staff to publish material arising from research as they think fit. In cases where commercial exploitation of the results is a possibility, however, the University requires that publication be withheld until appropriate protection can be put in place.

Employees Working at Other Institutions

- 2.6 University staff are frequently offered honorary appointments within other institutes and universities. In such circumstances, University staff should ensure through Legal Risk & Compliance that an appropriate agreement is in place between the University and the host institution for IP ownership and exploitation.

Employees Working at Partner NHS Trusts

- 2.7 University staff with clinical duties may also hold *honorary* contracts of employment with the relevant NHS partner trust. In such circumstances the ownership of IP will remain with the University. However, it should be noted the University has agreed specific arrangements with partner trusts for IP identification, ownership and exploitation. Further guidelines can be sought from the IP Manager within Partnership & Innovation.

Non-employees

- 2.8 The University offers honorary or visiting professorial positions to individuals who are not employed by the University. In such cases, individuals are usually required to transfer to the University any IP they create in the course of their honorary activities for the University, as if they were a University employee. Given the wider benefits such appointments bring to the University some discretion will, however, be allowed and agreement about ownership and exploitation of IP should be reached (through Human Resources and the IP Manager) at the time of appointment by the University.

Students

- 2.9 As part of the registration process students assign to the University any commercially-exploitable IP which they generate as a consequence of their studies or research, or which is created using University facilities. In assigning their ownership rights to the University, a student is accorded the same rights

as a member of staff, e.g. with respect to revenue sharing. It is recognised that where a student is sponsored by a third party, the terms of that sponsorship may override this position, and require the student to assign IP to the sponsoring organisation.

- 2.10 Students are able and encouraged to publish their research work in journals or dissertations, with agreement of their course director or research supervisor and subject to any appropriate prior IP protection.
- 2.11 The policy applies to students registered at the University (including any registered as University students as part of agreements with other institutions or through on-line programmes), whoever is their supervisor, course director or equivalent.
- 2.12 For the avoidance of doubt, the University does not seek to interfere in students' rights insofar as they relate to free use of lecture notes that may or may not be subject to copyright or IP generated outside their studies or research.
- 2.13 Students registered elsewhere but whose supervisors include University employees are not subject to this policy.

Third Parties

- 2.14 The University frequently enters into agreements with third parties which specify ownership of IP generated by University staff and students in research collaborations. In such cases IP may be owned by the University, the third party or jointly. In the majority of cases a lead party is mutually agreed before the collaboration is undertaken. If the collaboration is likely to produce commercially valuable IP, terms to be agreed in respect of IP will be through the Department of Legal, Risk & Compliance and may from time to time involve Partnerships & Innovation.

3. GOVERNANCE

Liverpool Intellectual Property

- 3.1 The University exploits IP in a variety of ways but the preferred route is through the Liverpool Intellectual Property (LIP) unit within the department of Partnerships & Innovation. LIP was established in order to generate the maximum societal and commercial impact of University activities through an academic focussed process of IP identification, evaluation, management and exploitation. It is the responsibility of LIP to maximise the commercial exploitation of University IP; however, it is recognised that such exploitation should not prejudice the University's core activities of research, teaching and Knowledge Exchange. LIP should also take into account the wider strategies and policies of the University in determining the exploitation route.

Operations Team

- 3.2 The Operations Team (“the Operations Team”) is the LIP working group responsible for the operational oversight and monitoring of the services provided by the Commercial Partner. It meets on a monthly basis.

Investment Panel

- 3.3 The Investment Panel (“the Panel”) is the group responsible for consideration of proposals to establish spin-out companies. It meets at the request of the Operations Team.

Commercial Board

- 3.4 The Commercial Board (“the Board”) is the group responsible for the strategic oversight of the services provided by LIP and the CP. The Board, which has a lay chair and a mixture of lay and University members, reports to the University’s Planning and Resources Committee. The Board operates under an appropriate scheme of delegation granted by the Committee and meets on a quarterly basis.

4. COMMERCIAL PARTNER

Appointment of a Partner

- 4.1 In September 2011 the University procured the IP management services of a Commercial Partner (CP) to aid the assessment of IP generated by the University through its research and other activities. The LIP and CP will work seamlessly to deliver services to University staff and students. Under the terms of the Consultancy Agreement the CP will provide IP exploitation services. Any disclosure made to the CP by the University will be held Confidential and will not affect the ownership rights to the IP. The University retains all the rights to the ownership of the IP and none will be diluted by the CP.

Timescales

- 4.2 The Operations Team will usually make a decision as to whether to support and, where appropriate, formally protect the disclosure, for example by filing for patent protection, within 8 weeks of disclosing the invention to the CP.
- 4.3 The timescale for exploitation of any given set of IP depends significantly on market conditions as well as the state of development of the IP. The discussions between LIP, the Inventor, and the CP should agree a reasonable initial timetable with regular review points. All parties involved have responsibilities in achieving successful exploitation and need to be aware that circumstances will change as the process develops. Exploitation of IP might

take the form of licensing, assignment or creation of a spin-out or start-up company.

5. EXTERNALLY-FUNDED RESEARCH PROJECTS

- 5.1 As well as disclosures from staff and students, LIP and the CP will monitor the University's externally-funded research projects to attempt to identify suitable opportunities. LIP will ensure that such monitoring is, where appropriate, carried out in conjunction with Business Gateway, the Research Support Office, the Contracts Office and academic staff to ensure consistency of approach, particularly if the monitoring involves discussions with external funders of research.

6. FURTHER DEVELOPMENT OF IP

- 6.1 It is recognised that it is often difficult to secure funding to develop research commercially as this often falls outside the funder's remit. Early development work can be the biggest hurdle to securing future commercial interest and funding for projects. In view of this, the University is seeking to establish separate funding to support those projects with the highest potential.

Investment Fund

- 6.2 There are a number of ways in which IP can be developed further to make it attractive to potential investors or customers and to improve the return for the University and the Inventors. The University intends to establish an Investment Fund to support the development of specific commercial activities. The Fund will be administered by the LIP and investment decisions will be taken by the Planning and Resources Committee following recommendations from the Board, in conjunction with relevant individuals from the University, e.g. Heads of School/Department in respect of an Inventor's time and any other resources required to undertake the project. In reaching a decision to invest in a project through the Investment Fund the University will need to be convinced not only that the project merits support, because of its commercial promise, but also that all other potential sources of external funding had been explored.

Commercial Activity Fund

- 6.3 There are a number of ways in which non-patent protectable opportunities can be developed further and exploited. The University also intends to establish its own Commercial Activity Fund (CAF). This will be administered by the LIP and investment decisions will be taken by the Planning and Resources Committee following recommendations from the Operations Team, in conjunction with relevant individuals from the University, e.g. Heads of School/Department in respect of an Inventor's time and any other resources required to undertake the project. Distribution of returns from non-patentable

commercial projects (for example, service based activities) will be agreed by the Head of School and considered in the light of the individual circumstances. Where the commercial activities are successful, following receipt of CAF investment, then the CAF investment will be repaid in full plus 15%.

7. COMMERCIALISATION THROUGH SPIN-OUT OR START-UP COMPANIES

- 7.1 Where exploitation involves the creation of a spin-out company (i.e. a company formed as a consequence of University IP) or a start-up company (a company created by a third party in which the University is invited to participate) the involvement of the Inventor in providing ongoing or future services to that company must be governed by an appropriate agreement, e.g. for the provision of consultancy or research services. Legal, Risk & Compliance will be responsible for drafting, reviewing and completing such agreements for the University.

Inventor(s) services

- 7.2 Inventor(s) are permitted to provide services (either as a director of the company or simply as an advisor) subject to the University's normal policy on consultancy and outside work activities. (See the University's Consultancy Services/CPD Policy). Inventors may also be seconded to the company subject to the normal contracting and approval processes, and in particular that it is subject in all cases to the full recovery of cost (including the use of University facilities and services). In general, such companies should not operate from University premises, except under a specific licence to occupy which would normally be in a defined incubator space.

Conflicts of Interest

- 7.3 The University's Conflicts of Interest Policy should be complied with in all circumstances. In particular, staff specifically employed to further the University's commercial activities are not normally permitted to acquire equity in spin-out companies whilst remaining as an employee of the University.

8. COMPANY DIRECTORSHIPS

- 8.1 Inventors are permitted to accept appointment to directorships in spin-out or start-up companies, subject to the agreement of their Head of School/Department and the Commercial Board. Individuals undertaking such roles should note that they are personally liable as a director and should ensure that they fully understand the legal responsibilities involved. In general, staff in senior positions and business development roles within the University should not undertake directorships as they may be conflicted with their University position.

9. DISTRIBUTION OF BENEFITS ARISING FROM EXPLOITATION

9.1 Returns from the commercialisation of University IP belong to the University. The University has a policy of granting a generous share of the returns to provide an incentive to the Inventors. The University only seeks to retain more of the returns based on its ownership of the IP if such returns are so substantial that it would be inappropriate as a charitable organisation for them not to be reinvested in the University's charitable objects.

Income Arising from IP Licensing, Selling or Assignment

9.2 The University is responsible for collecting income from third parties in respect of exploitation of IP. Where IP has been sold, assigned or licensed, this would usually take the form of milestone and/or royalty payments. As a guide to income distribution, after recovery of eligible costs such as external legal fees and distribution of any revenue sharing with the funders of the original research or third party collaborators, the surplus balance is distributed 80% to the Inventor(s) and 20% to the School of the Inventors for the first £100k and 50% Inventor(s), 20% Faculty and 30% the University over and above £100k.

Principles for Allocation of Income

9.3 The exact allocation of income will be determined by the Commercial Board according to the following principles:

- whether the invention was made in the course of normal duties
- whether the circumstances were such that an invention might reasonably be expected to result from the carrying out of those duties
- whether, because of their special responsibilities, the Inventor had a special obligation to further the interests of the University
- the nature of his/her duties, the remuneration and other advantages which the Inventor has derived from their position with the University
- the effort and skill which the Inventor has devoted to making the invention
- the extent to which the invention was made jointly by the Inventor with any other person and the effort and skill which such other person has devoted to the invention

Payment of Income

9.4 The funds allocated to the Inventor(s) may, at the discretion of the Inventor, take the form of a personal payment (in which case payment will be made via the University payroll and cover the University's National Insurance contribution) or be paid into a nominated University account.

Determining the Inventor

- 9.5 Many people may be involved with the work that leads up to the creation of IP and the work that is subsequently undertaken to exploit the IP. However, as a result of IP law, many of those involved will not actually be an Inventor as they have not been involved at the actual point of creation.
- 9.6 Where more than one Inventor has created the IP the Inventors may decide between themselves as to how they will share the Inventors' distributed funds. Where the Inventors cannot agree as to how to share the Inventors' distributed funds, then after further consultation with the CP to establish the inventive contribution of all Inventors, the Operations Team will decide.

Allocation of Equity within a Spin-out or Company Formation

- 9.7 In those cases where the consideration for commercialisation of University IP is equity (e.g. as part of a company formation) this will be allocated on the basis of 60% to University and 40% to the Inventor(s) who will usually hold such equity in a personal capacity. Inventor(s) should note that they will be liable for any relevant personal taxes in such holdings and they are advised to seek independent legal and financial advice. The University will identify and appoint an appropriate representation on the board.

Mixture of Equity and Royalties

- 9.8 Where the University receives a mixture of equity and royalties as payment for IP, the sum of both will be combined to meet the overall allocation to the Inventor as described above.

10. ADDITIONAL CONSIDERATION

- 10.1 Where the Operations Team have decided that an opportunity is non-patentable, or where the LIP and the CP have been unable to commercialise all or any part of the IP, then upon request from the Inventor(s), the University may, subject to any previous term or conditions assign all its rights, title and interest in such IP to the Inventor(s) in return for a perpetual non-exclusive royalty-free licence with the right to sub-liscence.
- 10.2 Where the Inventor(s) successfully commercialise any IP created by them, following assignment of such IP to them, then the Inventor(s) will reimburse any expenses previously incurred by the University in connection with protection of such IP, including legal costs and a 10% share of any income generated by or on behalf of the Inventor(s) from the commercialisation of such IP.

(Approved by the University of Liverpool Council on 11 January 2012)

F.7 Included Works

Despite them being present in the repo, we did not write or create the following:

- writeup/latex/tikz-uml.sty
- writeup/latex/todonotes.sty
- writeup/latex/ulem.sty
- tikz-styles.sty
- writeup/images/appendices/licence.png (CC0 licence logo)
- The text of any legal licence
- client/web_interface_mockup/jquery.js
- client/web_interface_mockup/turtles.ttf

F.8 The turtle image used as a favicon and in the banner

This was dedicated to the public domain by its creator. The UK both is a signatory of the berne convention and has precedent holding that licencing cannot be retroactivly rescinded.

F.9 jquery.js Licence

Copyright 2014 jQuery Foundation and other contributors <http://jquery.com/>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

F.10 todonotes.sty licence

Copyright (c) 2008 by Henrik Skov Midtiby <henrikmidtiby@gmail.com>

This file may be distributed and/or modified under the conditions of the LaTeX Project Public License, either version 1.2 of this license or (at your option) any later version. This licence is provided below.

F.11 The LaTeX Project Public License

LPPL Version 1.3c 2008-05-04

Copyright 1999 2002–2008 LaTeX3 Project

Everyone is allowed to distribute verbatim copies of this
license document, but modification of it is not allowed.

F.11.1 PREAMBLE

The LaTeX Project Public License (LPPL) is the primary license under which the LaTeX kernel and the base LaTeX packages are distributed.

You may use this license for any work of which you hold the copyright and which you wish to distribute. This license may be particularly suitable if your work is TeX-related (such as a LaTeX package), but it is written in such a way that you can use it even if your work is unrelated to TeX.

The section ‘WHETHER AND HOW TO DISTRIBUTE WORKS UNDER THIS LICENSE’, below, gives instructions, examples, and recommendations for authors who are considering distributing their works under this license.

This license gives conditions under which a work may be distributed and modified, as well as conditions under which modified versions of that work may be distributed.

We, the LaTeX3 Project, believe that the conditions below give you the freedom to make and distribute modified versions of your work that conform with whatever technical specifications you wish while maintaining the availability, integrity, and reliability of that work. If you do not see how to achieve your goal while meeting these conditions, then read the document ‘cfgguide.tex’ and ‘modguide.tex’ in the base LaTeX distribution for suggestions.

F.11.2 DEFINITIONS

In this license document the following terms are used:

‘Work’ Any work being distributed under this License.

‘Derived Work’ Any work that under any applicable law is derived from the Work.

'Modification' Any procedure that produces a Derived Work under any applicable law – for example, the production of a file containing an original file associated with the Work or a significant portion of such a file, either verbatim or with modifications and/or translated into another language.

'Modify' To apply any procedure that produces a Derived Work under any applicable law.

'Distribution' Making copies of the Work available from one person to another, in whole or in part. Distribution includes (but is not limited to) making any electronic components of the Work accessible by file transfer protocols such as FTP or HTTP or by shared file systems such as Sun's Network File System (NFS).

'Compiled Work' A version of the Work that has been processed into a form where it is directly usable on a computer system. This processing may include using installation facilities provided by the Work, transformations of the Work, copying of components of the Work, or other activities. Note that modification of any installation facilities provided by the Work constitutes modification of the Work.

'Current Maintainer' A person or persons nominated as such within the Work. If there is no such explicit nomination then it is the 'Copyright Holder' under any applicable law.

'Base Interpreter' A program or process that is normally needed for running or interpreting a part or the whole of the Work.

A Base Interpreter may depend on external components but these are not considered part of the Base Interpreter provided that each external component clearly identifies itself whenever it is used interactively. Unless explicitly specified when applying the license to the Work, the only applicable Base Interpreter is a 'LaTeX-Format' or in the case of files belonging to the 'LaTeX-format' a program implementing the 'TeX language'.

F.11.3 CONDITIONS ON DISTRIBUTION AND MODIFICATION

1. Activities other than distribution and/or modification of the Work are not covered by this license; they are outside its scope. In particular, the act of running the Work is not restricted and no requirements are made concerning any offers of support for the Work.
2. You may distribute a complete, unmodified copy of the Work as you received it. Distribution of only part of the Work is considered modification of the Work, and no right to distribute such a Derived Work may be assumed under the terms of this clause.
3. You may distribute a Compiled Work that has been generated from a complete, unmodified copy of the Work as distributed under Clause 2 above, as long as that Compiled Work is

distributed in such a way that the recipients may install the Compiled Work on their system exactly as it would have been installed if they generated a Compiled Work directly from the Work.

4. If you are the Current Maintainer of the Work, you may, without restriction, modify the Work, thus creating a Derived Work. You may also distribute the Derived Work without restriction, including Compiled Works generated from the Derived Work. Derived Works distributed in this manner by the Current Maintainer are considered to be updated versions of the Work.
5. If you are not the Current Maintainer of the Work, you may modify your copy of the Work, thus creating a Derived Work based on the Work, and compile this Derived Work, thus creating a Compiled Work based on the Derived Work.
6. If you are not the Current Maintainer of the Work, you may distribute a Derived Work provided the following conditions are met for every component of the Work unless that component clearly states in the copyright notice that it is exempt from that condition. Only the Current Maintainer is allowed to add such statements of exemption to a component of the Work.
 - a. If a component of this Derived Work can be a direct replacement for a component of the Work when that component is used with the Base Interpreter, then, wherever this component of the Work identifies itself to the user when used interactively with that Base Interpreter, the replacement component of this Derived Work clearly and unambiguously identifies itself as a modified version of this component to the user when used interactively with that Base Interpreter.
 - b. Every component of the Derived Work contains prominent notices detailing the nature of the changes to that component, or a prominent reference to another file that is distributed as part of the Derived Work and that contains a complete and accurate log of the changes.
 - c. No information in the Derived Work implies that any persons, including (but not limited to) the authors of the original version of the Work, provide any support, including (but not limited to) the reporting and handling of errors, to recipients of the Derived Work unless those persons have stated explicitly that they do provide such support for the Derived Work.
 - d. You distribute at least one of the following with the Derived Work:
 - (a) A complete, unmodified copy of the Work; if your distribution of a modified component is made by offering access to copy the modified component from a designated place, then offering equivalent access to copy the Work from the same or some similar place meets this condition, even though third parties are not compelled to copy the Work along with the modified component;

- (b) Information that is sufficient to obtain a complete, unmodified copy of the Work.
7. If you are not the Current Maintainer of the Work, you may distribute a Compiled Work generated from a Derived Work, as long as the Derived Work is distributed to all recipients of the Compiled Work, and as long as the conditions of Clause 6, above, are met with regard to the Derived Work.
 8. The conditions above are not intended to prohibit, and hence do not apply to, the modification, by any method, of any component so that it becomes identical to an updated version of that component of the Work as it is distributed by the Current Maintainer under Clause 4, above.
 9. Distribution of the Work or any Derived Work in an alternative format, where the Work or that Derived Work (in whole or in part) is then produced by applying some process to that format, does not relax or nullify any sections of this license as they pertain to the results of applying that process.
 10.
 - a. A Derived Work may be distributed under a different license provided that license itself honors the conditions listed in Clause 6 above, in regard to the Work, though it does not have to honor the rest of the conditions in this license.
 - b. If a Derived Work is distributed under a different license, that Derived Work must provide sufficient documentation as part of itself to allow each recipient of that Derived Work to honor the restrictions in Clause 6 above, concerning changes from the Work.
 11. This license places no restrictions on works that are unrelated to the Work, nor does this license place any restrictions on aggregating such works with the Work by any means.
 12. Nothing in this license is intended to, or may be used to, prevent complete compliance by all parties with all applicable laws.

F.11.4 NO WARRANTY

There is no warranty for the Work. Except when otherwise stated in writing, the Copyright Holder provides the Work ‘as is’, without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the Work is with you. Should the Work prove defective, you assume the cost of all necessary servicing, repair, or correction.

In no event unless required by applicable law or agreed to in writing will The Copyright Holder, or any author named in the components of the Work, or any other party who may distribute and/or modify the Work as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of any use of the Work or out of inability to use

the Work (including, but not limited to, loss of data, data being rendered inaccurate, or losses sustained by anyone as a result of any failure of the Work to operate with any other programs), even if the Copyright Holder or said author or said other party has been advised of the possibility of such damages.

F.11.5 MAINTENANCE OF THE WORK

The Work has the status ‘author-maintained’ if the Copyright Holder explicitly and prominently states near the primary copyright notice in the Work that the Work can only be maintained by the Copyright Holder or simply that it is ‘author-maintained’.

The Work has the status ‘maintained’ if there is a Current Maintainer who has indicated in the Work that they are willing to receive error reports for the Work (for example, by supplying a valid e-mail address). It is not required for the Current Maintainer to acknowledge or act upon these error reports.

The Work changes from status ‘maintained’ to ‘unmaintained’ if there is no Current Maintainer, or the person stated to be Current Maintainer of the work cannot be reached through the indicated means of communication for a period of six months, and there are no other significant signs of active maintenance.

You can become the Current Maintainer of the Work by agreement with any existing Current Maintainer to take over this role.

If the Work is unmaintained, you can become the Current Maintainer of the Work through the following steps:

1. Make a reasonable attempt to trace the Current Maintainer (and the Copyright Holder, if the two differ) through the means of an Internet or similar search.
2. If this search is successful, then enquire whether the Work is still maintained.
 - a. If it is being maintained, then ask the Current Maintainer to update their communication data within one month.
 - b. If the search is unsuccessful or no action to resume active maintenance is taken by the Current Maintainer, then announce within the pertinent community your intention to take over maintenance. (If the Work is a LaTeX work, this could be done, for example, by posting to comp.text.tex.)
3.
 - a. If the Current Maintainer is reachable and agrees to pass maintenance of the Work to you, then this takes effect immediately upon announcement.
 - b. If the Current Maintainer is not reachable and the Copyright Holder agrees that maintenance of the Work be passed to you, then this takes effect immediately upon announcement.

4. If you make an ‘intention announcement’ as described in 2b. above and after three months your intention is challenged neither by the Current Maintainer nor by the Copyright Holder nor by other people, then you may arrange for the Work to be changed so as to name you as the (new) Current Maintainer.
5. If the previously unreachable Current Maintainer becomes reachable once more within three months of a change completed under the terms of 3b) or 4), then that Current Maintainer must become or remain the Current Maintainer upon request provided they then update their communication data within one month.

A change in the Current Maintainer does not, of itself, alter the fact that the Work is distributed under the LPPL license.

If you become the Current Maintainer of the Work, you should immediately provide, within the Work, a prominent and unambiguous statement of your status as Current Maintainer. You should also announce your new status to the same pertinent community as in 2b) above.

F.11.6 WHETHER AND HOW TO DISTRIBUTE WORKS UNDER THIS LICENSE

This section contains important instructions, examples, and recommendations for authors who are considering distributing their works under this license. These authors are addressed as ‘you’ in this section.

Choosing This License or Another License

If for any part of your work you want or need to use *distribution* conditions that differ significantly from those in this license, then do not refer to this license anywhere in your work but, instead, distribute your work under a different license. You may use the text of this license as a model for your own license, but your license should not refer to the LPPL or otherwise give the impression that your work is distributed under the LPPL.

The document ‘modguide.tex’ in the base LaTeX distribution explains the motivation behind the conditions of this license. It explains, for example, why distributing LaTeX under the GNU General Public License (GPL) was considered inappropriate. Even if your work is unrelated to LaTeX, the discussion in ‘modguide.tex’ may still be relevant, and authors intending to distribute their works under any license are encouraged to read it.

A Recommendation on Modification Without Distribution

It is wise never to modify a component of the Work, even for your own personal use, without also meeting the above conditions for distributing the modified component. While you might intend

that such modifications will never be distributed, often this will happen by accident – you may forget that you have modified that component; or it may not occur to you when allowing others to access the modified version that you are thus distributing it and violating the conditions of this license in ways that could have legal implications and, worse, cause problems for the community. It is therefore usually in your best interest to keep your copy of the Work identical with the public one. Many works provide ways to control the behavior of that work without altering any of its licensed components.

How to Use This License

To use this license, place in each of the components of your work both an explicit copyright notice including your name and the year the work was authored and/or last substantially modified. Include also a statement that the distribution and/or modification of that component is constrained by the conditions in this license.

Here is an example of such a notice and statement:

```
%% pig.dtx
%% Copyright 2005 M. Y. Name
%
% This work may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either version 1.3
% of this license or (at your option) any later version.
% The latest version of this license is in
%   http://www.latex-project.org/lppl.txt
% and version 1.3 or later is part of all distributions of LaTeX
% version 2005/12/01 or later.
%
% This work has the LPPL maintenance status 'maintained'.
%
% The Current Maintainer of this work is M. Y. Name.
%
% This work consists of the files pig.dtx and pig.ins
% and the derived file pig.sty.
```

Given such a notice and statement in a file, the conditions given in this license document would apply, with the ‘Work’ referring to the three files ‘pig.dtx’, ‘pig.ins’, and ‘pig.sty’ (the last being generated from ‘pig.dtx’ using ‘pig.ins’), the ‘Base Interpreter’ referring to any ‘LaTeX-Format’, and both ‘Copyright Holder’ and ‘Current Maintainer’ referring to the person ‘M. Y. Name’.

If you do not want the Maintenance section of LPPL to apply to your Work, change ‘main-

tained' above into 'author-maintained'. However, we recommend that you use 'maintained', as the Maintenance section was added in order to ensure that your Work remains useful to the community even when you can no longer maintain and support it yourself.

Derived Works That Are Not Replacements

Several clauses of the LPPL specify means to provide reliability and stability for the user community. They therefore concern themselves with the case that a Derived Work is intended to be used as a (compatible or incompatible) replacement of the original Work. If this is not the case (e.g., if a few lines of code are reused for a completely different task), then clauses 6b and 6d shall not apply.

Important Recommendations

Defining What Constitutes the Work

The LPPL requires that distributions of the Work contain all the files of the Work. It is therefore important that you provide a way for the licensee to determine which files constitute the Work. This could, for example, be achieved by explicitly listing all the files of the Work near the copyright notice of each file or by using a line such as:

```
% This work consists of all files listed in manifest.txt.
```

in that place. In the absence of an unequivocal list it might be impossible for the licensee to determine what is considered by you to comprise the Work and, in such a case, the licensee would be entitled to make reasonable conjectures as to which files comprise the Work.

F.12 CC0 licence and licence.png licence

The CC0 licence and the licence.png image is licenced under the creative commons attribution international 4.0 licence, this is provided below (NB: This licence is licenced under itself.)

F.13 Creative Commons Attribution 4.0 International Public License

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution 4.0 International Public License ("Public License"). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

F.13.1 Section 1 - Definitions.

- a. **Adapted Material** means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.
- b. **Adapter's License** means the license You apply to Your Copyright and Similar Rights in Your contributions to Adapted Material in accordance with the terms and conditions of this Public License.
- c. **Copyright and Similar Rights** means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights.
- d. **Effective Technological Measures** means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.
- e. **Exceptions and Limitations** means fair use, fair dealing, and/or any other exception or limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.
- f. **Licensed Material** means the artistic or literary work, database, or other material to which the Licensor applied this Public License.
- g. **Licensed Rights** means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.
- h. **Licensor** means the individual(s) or entity(ies) granting rights under this Public License.
- i. **Share** means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.

- j. **Sui Generis Database Rights** means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.
- k. **You** means the individual or entity exercising the Licensed Rights under this Public License.
Your has a corresponding meaning.

F.13.2 Section 2 - Scope.

a. License grant .

- 1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:
 - A. reproduce and Share the Licensed Material, in whole or in part; and
 - B. produce, reproduce, and Share Adapted Material.
- 2. **Exceptions and Limitations.** For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.
- 3. **Term.** The term of this Public License is specified in Section 6(a).
- 4. **Media and formats; technical modifications allowed.** The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures. For purposes of this Public License, simply making modifications authorized by this Section 2(a)(4) never produces Adapted Material.
- 5. **Downstream recipients.** **A. Offer from the Licensor - Licensed Material.** Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.
B. No downstream restrictions. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.

6. **No endorsement.** Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).
- b. **Other rights.**
 1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.
 2. Patent and trademark rights are not licensed under this Public License.
 3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties.

F.13.3 Section 3 - License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. Attribution .

1. If You Share the Licensed Material (including in modified form), You must:
 - A. retain the following if it is supplied by the Licensor with the Licensed Material:
 - i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);
 - ii. a copyright notice;
 - iii. a notice that refers to this Public License;
 - iv. a notice that refers to the disclaimer of warranties;
 - v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;
 - B. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and
 - C. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.
2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it

may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.

3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.
4. If You Share Adapted Material You produce, the Adapter's License You apply must not prevent recipients of the Adapted Material from complying with this Public License.

F.13.4 Section 4 - Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

- a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database;
- b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material; and
- c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

F.13.5 Section 5 - Disclaimer of Warranties and Limitation of Liability.

- a. Unless otherwise separately undertaken by the Licensor, to the extent possible, the Licensor offers the Licensed Material as-is and as-available, and makes no representations or warranties of any kind concerning the Licensed Material, whether express, implied, statutory, or other. This includes, without limitation, warranties of title, merchantability, fitness for a particular purpose, non-infringement, absence of latent or other defects, accuracy, or the presence or absence of errors, whether or not known or discoverable. Where disclaimers of warranties are not allowed in full or in part, this disclaimer may not apply to You.
- b. To the extent possible, in no event will the Licensor be liable to You on any legal theory (including, without limitation, negligence) or otherwise for any direct, special, indirect, incidental, consequential, punitive, exemplary, or other losses, costs, expenses, or damages arising out of this Public License or use of the Licensed Material, even if the Licensor has been advised of the possibility of such losses, costs, expenses, or damages. Where a limitation of liability is not allowed in full or in part, this limitation may not apply to You.

- c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

F.13.6 Section 6 - Term and Termination.

- a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.
- b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:
 1. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or
 2. upon express reinstatement by the Licensor.

For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

- c. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.
- d. Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

F.13.7 Section 7 - Other Terms and Conditions.

- a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.
- b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

F.13.8 Section 8 - Interpretation.

- a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully be made without permission under this Public License.
- b. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the

provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.

- c. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.
- d. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.

F.14 ulem.sty licence

Copyright (c) 1989-2011 by Donald Arseneau (Vancouver, Canada; asnd@triumf.ca)

This software may be freely transmitted, reproduced, or modified for any purpose provided that this copyright notice is left intact. Small excerpts may be taken and used without any restriction.)

F.15 tikz-uml.sty licence

tikz-uml.sty is licenced under the LPPL, previously given.

F.16 tikz-styles.sty licence

tikz-styles.sty is licenced under the LPPL, previously given.

F.17 sqlite-jdbc-3.7.2.jar licence

sqlite-jdbc-3.7.2.jar is licenced under the Apache License version 2.0. Apache License

Version 2.0, January 2004

<http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

F.17.1 1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

F.17.2 2. Grant of Copyright License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work

and such Derivative Works in Source or Object form.

F.17.3 3. Grant of Patent License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

F.17.4 4. Redistribution.

You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

- a. You must give any other recipients of the Work or Derivative Works a copy of this License; and
- b. You must cause any modified files to carry prominent notices stating that You changed the files; and
- c. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
- d. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE

text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

F.17.5 5.

Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

F.17.6 6.

Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

F.17.7 7.

Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

F.17.8 8.

Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for

loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

F.17.9 9.

Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

F.18 winstone.jar licence

Winstone is licenced under the CDDL. COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0 1. Definitions. 1.1. "Contributor" means each individual or entity that creates or contributes to the creation of Modifications. 1.2. "Contributor Version" means the combination of the Original Software, prior Modifications used by a Contributor (if any), and the Modifications made by that particular Contributor. 1.3. "Covered Software" means (a) the Original Software, or (b) Modifications, or (c) the combination of files containing Original Software with files containing Modifications, in each case including portions thereof. 1.4. "Executable" means the Covered Software in any form other than Source Code. 1.5. "Initial Developer" means the individual or entity that first makes Original Software available under this License. 1.6. "Larger Work" means a work which combines Covered Software or portions thereof with code not governed by the terms of this License. 1.7. "License" means this document. 1.8. "Licensable" means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein. 1.9. "Modifications" means the Source Code and Executable form of any of the following: A. Any file that results from an addition to, deletion from or modification of the contents of a file containing Original Software or previous Modifications; B. Any new file that contains any part of the Original Software or previous Modification; or C. Any new file that is contributed or otherwise made available under the terms of this License. 1.10. "Original Software" means the Source Code and Executable form of computer software code that is originally released under this License. 1.11. "Patent Claims" means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor. 1.12. "Source Code" means (a) the common form of computer

software code in which modifications are made and (b) associated documentation included in or with such code.

1.13. "You" (or "Your") means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License. For legal entities, "You" includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50

F.19 CDDL licence licence

The CDDL is not the creation of BallmerPeak but was created by Sun Microsystems. The CDDL is licenced under CC Attribution 3.0 Unported. Fortunately CC-BY-3 allows merely linking to it, the licence can be found here: <http://creativecommons.org/licenses/by/3.0/legalcode>

F.20 Unlicenced Material

The following are reproduced and distributed without permission.

1. The text of the IP policy of The University of Liverpool

Appendix G

Source Listing

```
1 package ballmerpeak.turtlenet.server;
2
3 import java.util.Date;
4 import java.util.Scanner;
5
6 public class TNClient implements Runnable {
7     public NetworkConnection connection;
8     public Thread networkThread;
9     public Database db = null;
10    public String password = "NOT SET";
11    public boolean running = true;
12    public boolean dbReady = false;
13
14    public TNClient (String pw) {
15        password = pw;
16    }
17
18    public void run () {
19        if (!Crypto.keysExist())
20            Crypto.keyGen();
21
22        connection = new NetworkConnection("turtle.turtlenet.co.uk");
23        networkThread = new Thread(connection);
24        db = new Database(password);
25
26        networkThread.start();
27        dbReady = true;
28
29        while (running)
30            while (connection.hasMessage())
31                Parser.parse(Crypto.decrypt(connection.getMessage()), db);
32
33        connection.close();
34        db.dbDisconnect();
35        Logger.close();
36    }
37 }
```

```
1 package ballmerpeak.turtlenet.remoteserver;
2
3 import ballmerpeak.turtlenet.shared.Message;
4 import java.io.*;
5 import java.net.*;
6 import java.util.Date;
7 import java.util.StringTokenizer;
8 import javax.xml.bind.DatatypeConverter;
9
10 public class Server
11 {
12     public static String shutdownPassword = "SHUTDOWN 83eea84d472df09f5e64468996fdff0e";
13     private static ServerSocket socket;
14     private static boolean running = true;
15
16     public static void start (int port) {
17         Socket incoming;
18         Thread t;
19
20         try {
21             socket = new ServerSocket(port);
22
23             while (running) {
24                 incoming = socket.accept();
25                 t = new Thread(new Session(incoming));
26                 t.start();
27             }
28         } catch (Exception e) {
29             if (running)
30                 System.out.println("ERROR: " + e.getMessage());
31         } finally {
32             shutdown();
33         }
34     }
35
36     public static void shutdown() {
37         running = false;
38
39         try {
40             socket.close();
41         } catch (Exception e) {
42             System.out.println("ERROR: " + e.getMessage());
43             System.exit(1);
44         }
45     }
46
47     public static void main (String[] argv) {
48         System.out.println("Server running...");
49         start(31415);
50     }
51 }
52
53 class Session implements Runnable
54 {
55     private Socket client;
56
57     Session (Socket s) {
58         client = s;
59     }
60
61     // execute()s the clients command and then closes the connection.
62     public void run() {
63         System.out.println("Connection from " + client.getInetAddress().getHostAddress());
64         BufferedReader in = null;
65         PrintWriter out = null;
66
67         try {
68             in = new BufferedReader
69                 (new InputStreamReader(client.getInputStream()));
70             out = new PrintWriter
71                 (new OutputStreamWriter(client.getOutputStream()));
72
73             execute(in.readLine(), in, out);
74         } catch (IOException e) {
75             System.out.println("ERROR: " + e.getMessage());
76         }
77         out.flush();
78         //close everything related to this session
79         try {
80             in.close();
81         } catch (Exception e) {}
82
83         try {
84             out.close();
85         } catch (Exception e) {}
86
87         try {
88             client.close();
89         } catch (Exception e) {}
90     }
91
92     //Protocol:
93     //NB: The universe came into existance at midnight on january 1st 1970
```

```
94     // A typical session is the following:  
95     //   Connect -> Send command to server -> disconnect  
96     // Valid commands are the following:  
97     //   t           - request the number of milliseconds since midnight 1970-01-01  
98     //   s <string> - request that a string be stored on the server  
99     //   get <long> - get every message posted since <long> number of milliseconds past midnight 1970-01-01  
100    //   c <claim message> - claim a username UNENCRYPTED PUBLICALLY KNOWN  
101    // Responses are the following:  
102    //s           - success  
103    //e           - error  
104    //<long>     - number of milliseconds since midnight on 1970-01-01  
105    //<string>*  - (0 or more strings) messages requested using get  
106    public void execute(String cmd, BufferedReader in, PrintWriter out) {  
107        System.out.println("Received \" " + cmd + " \"");  
108  
109        if (cmd.equals(Server.shutdownPassword)) {  
110            System.out.println("WARNING: shutdown password should be loaded from config file");  
111            System.out.println("Shutting down");  
112            Server.shutdown();  
113        }  
114  
115        else if (cmd.equals("t")) {  
116            out.println(String.valueOf(new Date().getTime()));  
117            out.println("s");  
118        }  
119  
120        else if (cmd.length() > 2 && cmd.substring(0,1).equals("s")) {  
121            try {  
122                String message = cmd.substring(2);  
123                System.out.println("Storing: " + message);  
124                BufferedWriter writer = new BufferedWriter(  
125                    new FileWriter(  
126                        new File("./data/" + (new Date()).getTime() +  
127                            " " + Hasher.hash(message))));  
128                writer.write(message);  
129                writer.close();  
130                out.println("s");  
131            } catch (Exception e) {  
132                System.out.println("ERROR: Unable to save: " + e);  
133            }  
134        }  
135  
136        else if (cmd.length() > 4 && cmd.substring(0,3).equals("get")) {  
137            System.out.println(cmd);  
138            try {  
139                String timestamp = cmd.substring(4);  
140                long lastRead = Long.parseLong(timestamp);  
141  
142                File dataDir = new File("./data");  
143                File[] files = dataDir.listFiles();  
144                for (int i = 0; i < files.length; i++) {  
145                    if (lastRead <= getLastTimestamp(files[i])) {  
146                        BufferedReader reader = new BufferedReader(  
147                            new FileReader(files[i]));  
148                        String msg = reader.readLine();  
149                        out.println(msg);  
150                    }  
151                }  
152                out.println("s");  
153            } catch (Exception e) {  
154                System.out.println("ERROR: Cannot execute \" " + cmd + " \"");  
155                out.println("e");  
156            }  
157        }  
158  
159        else if (cmd.length() > 2 && cmd.substring(0,2).equals("c ")) {  
160            Message claim = Message.parse(  
161                new String(  
162                    DatatypeConverter.parseBase64Binary(  
163                        cmd.substring(2))));  
164  
165            String content = claim.getContent();  
166            File data = new File("./data/" + (new Date()).getTime() + " " + content);  
167            if(userExists(content)) {  
168                out.println("e");  
169            } else {  
170                try {  
171                    BufferedWriter writer = new BufferedWriter(new FileWriter(data));  
172                    writer.write(cmd);  
173                    writer.close();  
174                    out.println("s");  
175                } catch (Exception e) {  
176                    System.out.println("ERROR: Could not write claim to disk");  
177                    out.println("e");  
178                }  
179            }  
180        }  
181  
182        else {  
183            System.out.println("Received \" " + cmd + " \", ignoring it");  
184            out.println("e");  
185        }  
186    }
```

```
187         out.flush();
188     }
189
190     //44634633434 HASH -> 44634633434
191     private long getTimestamp (File f) {
192         try {
193             String fn = f.getName();
194             if (fn.indexOf("_") != -1) {
195                 String ts = fn.substring(0, fn.indexOf("_"));
196                 return Long.parseLong(ts);
197             }
198         } catch (Exception e) {
199             System.out.println("ERROR: Could not parse file timestamp: " + e);
200         }
201         return 1;
202     }
203
204     private Boolean userExists (String name) {
205         File dir = new File("./data");
206         File[] files = dir.listFiles();
207         for (int i = 0; i < files.length; i++) {
208             if (files[i].getName().indexOf(" ") != -1) {
209                 String fname = files[i].getName();
210                 String[] tokens = new String[2];
211                 StringTokenizer tokenizer = new StringTokenizer(fname, " ", false);
212                 tokens[0] = tokenizer.nextToken();
213                 tokens[1] = "";
214                 while (tokenizer.hasMoreTokens())
215                     tokens[1] += tokenizer.nextToken();
216                 if (tokens[1].equals(name))
217                     return true;
218             }
219         }
220     }
221     return false;
222 }
```

```
1  body {
2      margin: 0;
3      background: #F2F2F2;
4      color: #666666;
5      font-family: "Lucida Console", monospace;
6  }
7
8  h1 {
9      font-size: 5em;
10     margin: 0;
11     letter-spacing: 10px;
12     font-weight: lighter;
13     padding: 10px 0 0 0;
14     text-align: center;
15  }
16
17 #header {
18     width: 1000px;
19     height: 120px;
20     margin: 0 auto;
21     background: #E0EEEE;
22     font-family: Sans-Serif;
23     color: #61B329;
24  }
25
26 #loading {
27     width: 1000px;
28     margin: 0 auto;
29     text-align: center;
30     font-size: 1.25em;
31  }
32
33 .gwt-login {
34     width: 1000px;
35     margin: 0 auto;
36     background: #61B329;
37     text-align: center;
38     font-size: 0.75em;
39     color: #000000;
40  }
41
42 .gwt-navigation {
43     width: 1000px;
44     height: 70px;
45     margin: 0 auto;
46     background: #61B329;
47     text-align: center;
48     padding: 25px 75px 0 0;
49  }
50
51 .gwt-friends-list {
52     width: 1000px;
53     margin: 0 auto;
54     background: #E0EEEE;
55     color: #666666;
56  }
57
58 .gwt-post-panel {
59     width: 700px;
60     margin: 0 auto;
61     background: #E0EEEE;
62     color: #666666;
63     padding: 20px 100px 20px 200px;
64  }
65
66 .gwt-comments-contents {
67     width: 675px;
68     margin: 0 auto;
69     background: #E0EEEE;
70     color: #666666;
71     padding: 20px 0 20px 60px;
72  }
73
74 .gwt-comments {
75     width: 700px;
76     margin: 0 auto;
77     background: #E0EEEE;
78     color: #666666;
79  }
80
81 .gwt-wall {
82     width: 1000px;
83     margin: 0 auto;
84     background: #E0EEEE;
85     color: #666666;
86  }
87
88 .gwt-wall-control {
89     width: 1000px;
90     margin: 0 auto;
91     background: #E0EEEE;
92     color: #666666;
93     padding: 0 150px 0 200px;
```

```
94    }
95
96    .gwt-create-post {
97        width: 800px;
98        margin: 0 auto;
99        background: #E0EEEE;
100       color: #666666;
101      padding: 0 0 0 200px;
102    }
103
104   .gwt-post-contents-footer {
105        width: 700px;
106        margin: 0 auto;
107        background: #E0EEEE;
108        color: #666666;
109        padding: 0 100px 0 10px;
110    }
111
112   .gwt-edit-group {
113        width: 1000px;
114        margin: 0 auto;
115        background: #E0EEEE;
116        color: #666666;
117        padding: 0 300px 0 115px;
118    }
119
120   .gwt-new-group {
121        width: 1000px;
122        margin: 0 auto;
123        background: #E0EEEE;
124        color: #666666;
125        padding: 0 300px 0 200px;
126    }
127
128   .gwt-friend {
129        width: 1000px;
130        margin: 0 auto;
131        background: #E0EEEE;
132        color: #666666;
133        padding: 0 0 0 200px;
134    }
135
136   .gwt-conversation-list {
137        width: 1000px;
138        margin: 0 auto;
139        background: #E0EEEE;
140        color: #666666;
141        padding: 0 200px 0 50px;
142    }
143
144   .gwt-conversation {
145        width: 950px;
146        margin: 0 auto;
147        background: #E0EEEE;
148        color: #666666;
149        padding: 0 0 0 50px;
150    }
151
152   .gwt-my-details {
153        width: 1000px;
154        margin: 0 auto;
155        background: #E0EEEE;
156        color: #666666;
157        padding: 0 0 0 200px;
158    }
159
160   .gwt-my-details-permissions {
161        width: 1000px;
162        margin: 0 auto;
163        background: #E0EEEE;
164        color: #666666;
165        padding: 0 200px 0 200px;
166    }
167
168   .gwt-friends-details {
169        width: 1000px;
170        margin: 0 auto;
171        background: #E0EEEE;
172        color: #666666;
173        padding: 0 0 10px 200px;
174    }
175
176   .gwt-Anchor {
177        font-size: 1.2em;
178        font-weight: bold;
179        color: #333333;
180    }
181
182   .gwt-Anchor:visited {
183        color: #666666;
184    }
185
186   .gwt-Anchor:hover {
```

```
187     color: #666666;
188 }
189
190 .gwt-Button {
191     width: 150px;
192 }
193
194 .gwt-TextArea-readonly {
195     background-color: #EEEEEE;
196     color: #555555;
197 }
198
199 .gwt-TextBox-readonly {
200     background-color: #EEEEEE;
201     color: #555555;
202 }
```

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
2  <html xmlns="http://www.w3.org/1999/xhtml">
3      <head>
4          <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5          <title>Turtlenet</title>
6          <link href="frontend.css" rel="stylesheet" type="text/css" />
7          <link href="favicon.ico" rel="shortcut icon" type="image/x-icon" />
8          <link rel="icon" type="image/png" href="favicon.png" />
9          <script type="text/javascript" language="javascript" src="frontend/frontend.nocache.js"></script>
10     </head>
11
12     <body>
13         <iframe src="javascript:''" id="__gwt_historyFrame" tabIndex='-1'
14             style="position:absolute;width:0;height:0;border:0"></iframe>
15
16         <div id="header">
17             <h1></h1>
18         </div>
19
20         <div id = "loading">
21             <style type="text/css">
22                 #loading {
23                     padding: 20px 0 0 0;
24                 }
25             </style>
26             Loading Turtlenet...
27         </div>
28     </body>
29 </html>
```

```
1 package ballmerpeak.turtlenet.client;
2
3 import ballmerpeak.turtlenet.shared.CommentDetails;
4 import ballmerpeak.turtlenet.shared.PostDetails;
5 import ballmerpeak.turtlenet.shared.Message;
6 import ballmerpeak.turtlenet.shared.Conversation;
7
8 import com.google.gwt.core.client.*;
9 import com.google.gwt.event.dom.client.*;
10 import com.google.gwt.user.client.ui.*;
11 import com.google.gwt.event.logical.shared.*;
12 import com.google.gwt.user.client.Window;
13 import com.google.gwt.user.client.rpc.AsyncCallback;
14 import com.google.gwt.dom.client.Style.FontWeight;
15 import com.google.gwt.dom.client.DivElement;
16 import com.google.gwt.dom.client.Document;
17 import com.google.gwt.user.client.Timer;
18 import com.google.gwt.user.client.Window;
19 import java.util.Date;
20
21 public class frontend implements EntryPoint, ClickListener {
22
23     // Create remote service proxy to talk to the server-side Turtlenet service
24     private final TurtlenetAsync turtlenet = GWT.create(Turtlenet.class);
25     //private final TurtlenetAsync msgfactory = GWT.create(MessageFactory.class);
26     public void onModuleLoad() {
27         // Remove loading indicatior from frontend.html
28         DivElement loadingIndicator = DivElement.as(Document.get().getElementById("loading"));
29         loadingIndicator.setInnerHTML("");
30
31         /* Add handler for window closing */
32         Window.addCloseHandler(new CloseHandler<Window>() {
33             public void onClose(CloseEvent<Window> event) {
34                 turtlenet.stopTN(new AsyncCallback<String>() {
35                     public void onFailure(Throwable caught) {
36                         //pretend nothing happened
37                     }
38                     public void onSuccess(String result) {
39                         //bask in success
40                     }
41                 });
42             }
43         });
44
45         // Call method to load the initial login page
46         login();
47     }
48
49     private String location = new String("");
50     private String refreshID = new String("");
51
52     // LOUISTODO May need to remove ' = new FlexTable()'
53     private FlexTable loginPanel = new FlexTable();
54     private void login() {
55         location = "login";
56         refreshID = "";
57         RootPanel.get().clear();
58         loginPanel = new FlexTable();
59         loginPanel.clear();
60         RootPanel.get().add(loginPanel);
61
62         // Create login panel widgets
63         final Button loginButton = new Button("Login");
64         loginButton.addClickListener(this);
65         final PasswordTextBox passwordInput = new PasswordTextBox();
66         final Label passwordLabel = new Label();
67
68         turtlenet.isFirstTime(new AsyncCallback<String>() {
69             public void onFailure(Throwable caught) {
70                 System.out.println("turtlenet.isFirstTime failed: " + caught);
71             }
72             public void onSuccess(String result) {
73                 if(result.equals("true")) { //GWT can only return objects
74                     passwordLabel.setText("Please choose a password:");
75                     final PasswordTextBox passwordConfirmInput = new PasswordTextBox();
76                     final Label passwordConfirmLabel = new Label("");
77                     passwordConfirmLabel.setText("Confirm your password:");
78                     final TextBox usernameInput = new TextBox();
79                     final Label usernameLabel = new Label("");
80                     usernameLabel.setText("Please choose a username:");
81
82                     // Add widgets to login panel
83                     loginPanel.setWidget(1, 1, usernameLabel);
84                     loginPanel.setWidget(2, 1, usernameInput);
85                     loginPanel.setWidget(3, 1, passwordLabel);
86                     loginPanel.setWidget(4, 1, passwordInput);
87                     loginPanel.setWidget(5, 1, passwordConfirmLabel);
88                     loginPanel.setWidget(6, 1, passwordConfirmInput);
89                     loginPanel.setWidget(7, 1, loginButton);
90
91                     // Add click handler for button
92                     loginButton.addClickListener(new ClickHandler() {
93                         public void onClick(ClickEvent event) {
```

```
94     passwordLabel.setText("Please choose a password:");
95     passwordLabel.getElement().getStyle().setProperty("color", "#000000");
96     passwordConfirmLabel.setText("Confirm your password");
97     passwordConfirmLabel.getElement().getStyle().setProperty("color", "#000000");
98     usernameLabel.setText("Please choose a username:");
99     usernameLabel.getElement().getStyle().setProperty("color", "#000000");
100
101    if(usernameInput.getText().equals("")) {
102        usernameLabel.setText("Must enter a username");
103        usernameLabel.getElement().getStyle().setProperty("color", "#FFFF00");
104    } else if(passwordInput.getText().equals("")) {
105        passwordLabel.setText("Must enter a password");
106        passwordLabel.getElement().getStyle().setProperty("color", "#FFFF00");
107    } else if(passwordConfirmInput.getText().equals("")) {
108        passwordConfirmLabel.setText("Must confirm password");
109        passwordConfirmLabel.getElement().getStyle().setProperty("color", "#FFFF00");
110    } else if(passwordInput.getText().equals(passwordConfirmInput.getText())) {
111        turtlenet.register(usernameInput.getText(), passwordInput.getText(), new
112        AsyncCallback<String>() {
113            public void onFailure(Throwable caught) {
114                System.out.println("turtlenet.register failed: " + caught);
115            }
116            public void onSuccess(String result) {
117                if(result.equals("success")) {
118                    turtlenet.getMyKey(new AsyncCallback<String>() {
119                        public void onFailure(Throwable caught) {
120                            System.out.println("turtlenet.getMyKey failed: " + caught);
121                        }
122                        public void onSuccess(String result) {
123                            wall(result, false);
124                        }
125                    });
126                } else if(result.equals("taken")) {
127                    usernameLabel.setText("Username already taken. Try again:");
128                    usernameLabel.getElement().getStyle().setProperty("color", "#FFFF00");
129                } else {
130                    System.out.println("turtlenet.register onSucess String result did not equal
131                     success or taken");
132                }
133            });
134        } else {
135            passwordLabel.setText("Passwords do not match. Try again:");
136            passwordLabel.getElement().setProperty("color", "#FFFF00");
137            passwordConfirmInput.setText("");
138            passwordInput.setText("");
139        }
140    });
141
142    } else {
143        passwordLabel.setText("Please enter your password:");
144
145        // Add widgets to login panel
146        loginPanel.setWidget(1, 1, passwordLabel);
147        loginPanel.setWidget(2, 1, passwordInput);
148        loginPanel.setWidget(3, 1, loginButton);
149
150        // Add click handler for button
151        loginButton.addClickHandler(new ClickHandler() {
152            public void onClick(ClickEvent event) {
153                passwordLabel.setText("Please enter your password:");
154
155                turtlenet.startTN(passwordInput.getText(), new AsyncCallback<String>() {
156                    public void onFailure(Throwable caught) {
157                        System.out.println("turtlenet.startTN failed: " + caught);
158                    }
159                    public void onSuccess(String result) {
160                        if(result.equals("success")) {
161                            turtlenet.getMyKey(new AsyncCallback<String>() {
162                                public void onFailure(Throwable caught) {
163                                    System.out.println("turtlenet.getMyKey failed: " + caught);
164                                }
165                                public void onSuccess(String result) {
166                                    wall(result, false);
167                                }
168                            });
169                        } else if(result.equals("failure")) {
170                            passwordLabel.setText("Password incorrect. Try again:");
171                        } else {
172                            System.out.println("turtlenet.startTN onSuccess String does not equal success or
173                             failure");
174                            passwordLabel.setText("INVALID RESPONSE FROM TNClient");
175                        }
176                    });
177                });
178            });
179        });
180
181        // Add style name for CSS
182
183    }
```

```
184     loginPanel.addStyleName("gwt-login");
185 }
186
187 // Used to track the most recent wall post to be displayed
188 Long wallLastTimeStamp = 0L;
189 Long conversationLastTimeStamp = 0L;
190 Long commentsLastTimeStamp = 0L;
191
192 // When the login button is clicked we start a repeating timer that refreshes
193 // the page every 5 seconds.
194 public void onClick(Widget sender) {
195     Timer refresh = new Timer() {
196         public void run() {
197             if(location.equals("wall")) {
198                 turtlenet.timeMostRecentWallPost(refreshID, new AsyncCallback<Long>() {
199                     public void onFailure(Throwable caught) {
200                         System.out.println("turtlenet.timeMostRecentWallPost failed: " + caught);
201                     }
202                     public void onSuccess(Long result) {
203                         if(result > wallLastTimeStamp) {
204                             System.out.println("Refreshing wall. refreshID: " + refreshID);
205                             wall(refreshID, true);
206                         }
207                     }
208                 });
209             } else if(location.equals("conversationList")) {
210                 System.out.println("Refreshing conversationList");
211                 conversationList();
212             } else if(location.equals("conversation")) {
213                 turtlenet.getConvoLastUpdated(refreshID, new AsyncCallback<Long>() {
214                     public void onFailure(Throwable caught) {
215                         //TODO Error
216                     }
217                     public void onSuccess(Long result) {
218                         if(result > conversationLastTimeStamp) {
219                             System.out.println("Refreshing conversation. refreshID: " + refreshID);
220                             conversation(refreshID, true);
221                         }
222                     }
223                 });
224             } else {
225                 //Do nothing
226             }
227         }
228     };
229     refresh.scheduleRepeating(5*1000);
230 }
231
232 private void navigation() {
233     HorizontalPanel navigationPanel = new HorizontalPanel();
234     RootPanel.get().add(navigationPanel);
235
236     // Create navigation links
237     Anchor linkMyWall = new Anchor("My Wall");
238     linkMyWall.getElement().getStyle().setProperty("paddingLeft", "100px");
239     Anchor linkMyDetails = new Anchor("My Details");
240     linkMyDetails.getElement().getStyle().setProperty("paddingLeft", "100px");
241     Anchor linkConversations = new Anchor("Messages");
242     linkConversations.getElement().getStyle().setProperty("paddingLeft", "100px");
243     Anchor linkFriends = new Anchor("Friends");
244     linkFriends.getElement().getStyle().setProperty("paddingLeft", "100px");
245     Anchor linkLogout = new Anchor("Logout");
246     linkLogout.getElement().getStyle().setProperty("paddingLeft", "100px");
247
248     // Add links to navigation panel
249     navigationPanel.add(linkMyWall);
250     navigationPanel.add(linkMyDetails);
251     navigationPanel.add(linkConversations);
252     navigationPanel.add(linkFriends);
253     navigationPanel.add(linkLogout);
254
255     // Add style name for CSS
256     navigationPanel.addStyleName("gwt-navigation");
257
258     // Add click handlers for anchors
259     linkMyWall.addClickHandler(new ClickHandler() {
260         public void onClick(ClickEvent event) {
261             turtlenet.getKey(new AsyncCallback<String>() {
262                 public void onFailure(Throwable caught) {
263                     System.out.println("turtlenet.getKey failed: " + caught);
264                 }
265                 public void onSuccess(String result) {
266                     wall(result, false);
267                 }
268             });
269         }
270     });
271
272     // Add click handlers for anchors
273     linkMyDetails.addClickHandler(new ClickHandler() {
274         public void onClick(ClickEvent event) {
275             myDetails();
276         }
277     });
278 }
```

```
277     });
278 
279     linkConversations.addClickHandler(new ClickHandler() {
280         public void onClick(ClickEvent event) {
281             conversationList();
282 
283             System.out.println("Wake up, Neo...");
284         }
285     });
286 
287     linkFriends.addClickHandler(new ClickHandler() {
288         public void onClick(ClickEvent event) {
289             friendsList("All");
290         }
291     });
292 
293     linkLogout.addClickHandler(new ClickHandler() {
294         public void onClick(ClickEvent event) {
295             turtlenet.stopTN(new AsyncCallback<String>() {
296                 public void onFailure(Throwable caught) {
297                     System.out.println("turtlenet.stopTN failed: " + caught);
298                 }
299                 public void onSuccess(String result) {
300                     login();
301                 }
302             });
303         }
304     });
305 
306     String[][] friendsListCategoryMembers = new String[0][0];
307     String[][] friendsListCategoryList = new String[0][0];
308     private TextBox friendsListPanel_myKeyTextBox;
309     private void friendsList(final String currentGroupID) {
310         location = "friendsList";
311         refreshID = "";
312 
313         RootPanel.get().clear();
314         navigation();
315         final FlexTable friendsListPanel = new FlexTable();
316         RootPanel.get().add(friendsListPanel);
317 
318         // Column title for anchors linking to messages
319         Label friendsNameLabel = new Label("Friend's Name");
320         friendsNameLabel.getElement().getStyle().setFontWeight(FontWeight.BOLD);
321         friendsNameLabel.getElement().getStyle().setProperty("paddingLeft", "100px");
322         friendsListPanel.setWidget(1, 0, friendsNameLabel);
323 
324         // Column title for labels outputting the date a message was received
325         Label friendsKeyLabel = new Label("Friend's Public Key");
326         friendsKeyLabel.getElement().getStyle().setFontWeight(FontWeight.BOLD);
327         friendsListPanel.setWidget(1, 1, friendsKeyLabel);
328 
329         turtlenet.getCategoryMembers(currentGroupID, new AsyncCallback<String[][]>() {
330             int i;
331             public void onFailure(Throwable caught) {
332                 System.out.println("turtlenet.getCategoryMembers failed: " + caught);
333             }
334             public void onSuccess(String[][] _result) {
335                 friendsListCategoryMembers = _result;
336                 for (i = 0; i < friendsListCategoryMembers.length; i++) {
337                     //list names/keys
338                     Anchor linkFriendsWall = new Anchor(friendsListCategoryMembers[i][0]);
339                     linkFriendsWall.getElement().getStyle().setProperty("paddingLeft", "100px");
340                     friendsListPanel.setWidget((i + 2), 0, linkFriendsWall);
341                     final String resultString = friendsListCategoryMembers[i][1];
342                     TextBox friendKeyBox = new TextBox();
343                     friendKeyBox.setText(resultString);
344                     friendKeyBox.setVisibleLength(75);
345                     friendKeyBox.setReadOnly(true);
346                     friendsListPanel.setWidget((i + 2), 1, friendKeyBox);
347                     //link names to walls
348                     System.out.println("adding link to " + friendsListCategoryMembers[i][0] + "'s wall");
349                     final String fkey = friendsListCategoryMembers[i][1];
350                     linkFriendsWall.addClickHandler(new ClickHandler() {
351                         public void onClick(ClickEvent event) {
352                             wall(fkey, false);
353                         }
354                     });
355                 }
356             }
357         });
358 
359         int row = friendsListPanel.getRowCount() + 2;
360 
361         if(!currentGroupID.equals("All")) {
362             Label currentGroupLabel = new Label(currentGroupID);
363             friendsListPanel.setWidget((row - 1), 3, currentGroupLabel);
364         }
365 
366         final ListBox currentGroups = new ListBox();
367         currentGroups.setVisibleItemCount(1);
368         currentGroups.setWidth("150px");
```

```
369     currentGroups.addItem("All");
370     friendsListPanel.setWidget(3, 3, currentGroups);
371 
372     turtlenet.getCategories(new AsyncCallback<String[][]>() {
373         int i;
374         public void onFailure(Throwable caught) {
375             System.out.println("turtlenet.getCategories failed: " + caught);
376         }
377         int selected;
378         public void onSuccess(String[][] _result) {
379             friendsListCategoryList = _result;
380             for (i = 0; i < friendsListCategoryList.length; i++) {
381                 currentGroups.addItem(friendsListCategoryList[i][0]);
382                 // Check if the group we've just added is the current group
383                 // If it is note the index using selected. We need to add
384                 // 1 to selected as "All" always appears first in the list.
385                 if(friendsListCategoryList[i][0].equals(currentGroupID)) {
386                     selected = (i + 1);
387                 }
388             }
389             // Use selected to set the selected item in the listbox to the
390             // current group
391             currentGroups.setSelectedIndex(selected);
392 
393             currentGroups.addChangeHandler(new ChangeHandler() {
394                 public void onChange(ChangeEvent event) {
395                     friendsList(currentGroups.getItemText(currentGroups.getSelectedIndex()));
396                 }
397             });
398         });
399     });
400 
401     Button newGroup = new Button("Add new category");
402     friendsListPanel.setWidget(2, 3, newGroup);
403     newGroup.addClickHandler(new ClickHandler() {
404         public void onClick(ClickEvent event) {
405             newGroup();
406         }
407     });
408 
409     friendsListPanel_myKeyTextBox = new TextBox();
410     friendsListPanel_myKeyTextBox.setWidth("480px");
411     friendsListPanel_myKeyTextBox.setReadOnly(true);
412 
413     turtlenet.getMyKey(new AsyncCallback<String>() {
414         public void onFailure(Throwable caught) {
415             System.out.println("turtlenet.getMyKey failed: " + caught);
416         }
417         public void onSuccess(String result) {
418             friendsListPanel_myKeyTextBox.setText(result);
419         }
420     });
421 
422     Label myKeyLabel = new Label("My key:");
423     myKeyLabel.getElement().getStyle().setFontWeight(FontWeight.BOLD);
424     myKeyLabel.getElement().getStyle().setProperty("paddingLeft", "100px");
425     friendsListPanel.setWidget((row - 1), 0, myKeyLabel);
426     friendsListPanel.setWidget((row - 1), 1, friendsListPanel_myKeyTextBox);
427 
428     if(currentGroupID.equals("All")) {
429         Button addFriend = new Button("Add new friend");
430         friendsListPanel.setWidget(1, 3, addFriend);
431         addFriend.addClickHandler(new ClickHandler() {
432             public void onClick(ClickEvent event) {
433                 addFriend();
434             }
435         });
436     } else {
437         Button editGroup = new Button("Edit category");
438         friendsListPanel.setWidget(1, 3, editGroup);
439         editGroup.addClickHandler(new ClickHandler() {
440             public void onClick(ClickEvent event) {
441                 editGroup(currentGroupID);
442             }
443         });
444     }
445 
446     // Add style name for CSS
447     friendsListPanel.addStyleName("gwt-friends-list");
448 }
449 
450 FlexTable conversationListPanel;
451 private void conversationList() {
452     location = "conversationList";
453     refreshID = "";
454 
455     //Setup basic page
456     RootPanel.get().clear();
457     navigation();
458 
459     //Create panel to contain widgets
460     conversationListPanel = new FlexTable();
461     RootPanel.get().add(conversationListPanel);
```

```
462     turtlenet.getConversations(new AsyncCallback<Conversation[]>() {
463         Conversation[] result;
464         public void onFailure(Throwable caught) {
465             System.out.println("turtlenet.getConversations failed: " + caught);
466         }
467         public void onSuccess(Conversation[] _result) {
468             result = _result;
469             for(int j = 0; j < result.length; j++) {
470                 System.out.println(result[j]);
471             }
472             System.out.println("result.length = " + result.length);
473             for (int i = 0; i < result.length; i++) {
474                 final String conversationID = result[i].signature;
475                 // Substrings dont work if we set the end point so its
476                 // bigger than our string. If the length is less than 40
477                 // we output the full string. If the string is 40 or
478                 // about we take the first 40 characters and add ...
479                 String linkText = new String("");
480                 if ((result[i].firstMessage).length() < 40) {
481                     linkText = (result[i].firstMessage);
482                 } else {
483                     linkText = (result[i].firstMessage).substring(1, 40) + "...";
484                 }
485                 Anchor linkConversation = new Anchor(linkText);
486                 conversationListPanel.setWidget(i, 0, linkConversation);
487
488                 // Add click handlers for anchors
489                 linkConversation.addClickHandler(new ClickHandler() {
490                     public void onClick(ClickEvent event) {
491                         conversation(conversationID, false);
492                     }
493                 });
494                 Label conversationParticipants = new Label(result[i].concatNames());
495                 conversationListPanel.setWidget(i, 1, conversationParticipants);
496             }
497             Button newConversation = new Button("New conversation");
498             newConversation.setWidth("400px");
499             newConversation.addClickHandler(new ClickHandler() {
500                 public void onClick(ClickEvent event) {
501                     newConversation();
502                 }
503             });
504         });
505         conversationListPanel.setWidget((result.length + 1), 0, newConversation);
506     });
507 });
508 });
509
510 // Add style name for CSS
511 conversationListPanel.addStyleName("gwt-conversation-list");
512 }
513
514 private void myDetails() {
515     location = "myDetails";
516     refreshID = "";
517
518     RootPanel.get().clear();
519     navigation();
520     FlexTable myDetailsPanel = new FlexTable();
521     RootPanel.get().add(myDetailsPanel);
522
523     // Create widgets relating to username
524     Label usernameLabel = new Label("Username:");
525     myDetailsPanel.setWidget(0, 0, usernameLabel);
526
527     final TextBox editUsername = new TextBox();
528     editUsername.setWidth("300px");
529     turtlenet.getMyUsername(new AsyncCallback<String>() {
530         public void onFailure(Throwable caught) {
531             System.out.println("turtlenet.getMyUsername failed: " + caught);
532         }
533         public void onSuccess(String result) {
534             editUsername.setText(result);
535         }
536     });
537
538     myDetailsPanel.setWidget(0, 1, editUsername);
539
540     Button saveUsername = new Button("Save Username");
541     myDetailsPanel.setWidget(0, 2, saveUsername);
542
543     final Label editUsernameLabel = new Label();
544     editUsernameLabel.setWidth("200px");
545     myDetailsPanel.setWidget(0, 3, editUsernameLabel);
546
547     saveUsername.addClickHandler(new ClickHandler() {
548         public void onClick(ClickEvent event) {
549             turtlenet.claimUsername(editUsername.getText(), new AsyncCallback<String>() {
550                 public void onFailure(Throwable caught) {
551                     System.out.println("turtlenet.claimUsername failed: " + caught);
552                 }
553                 public void onSuccess(String result) {
554                     if (result.equals("success")) {
```

```
555             editUsernameLabel.setText("Username saved");
556         } else if (result.equals("failure")) {
557             editUsernameLabel.setText("Username taken");
558         }
559     });
560 }
561 });
562 });
563 }
564 // Create widgets relating to name
565 Label nameLabel = new Label("Name:");
566 myDetailsPanel.setWidget(1, 0, nameLabel);
567
568 final TextBox editName = new TextBox();
569 editName.setWidth("300px");
570 turtlenet.getMyPDATA("name", new AsyncCallback<String>() {
571     public void onFailure(Throwable caught) {
572         System.out.println("turtlenet.getMyPDATA name failed: " + caught);
573     }
574     public void onSuccess(String result) {
575         editName.setText(result);
576     }
577 });
578 myDetailsPanel.setWidget(1, 1, editName);
579
580 Button saveName = new Button("Save Name");
581 myDetailsPanel.setWidget(1, 2, saveName);
582
583 final Label editNameLabel = new Label();
584 myDetailsPanel.setWidget(1, 3, editNameLabel);
585
586 saveName.addClickHandler(new ClickHandler() {
587     public void onClick(ClickEvent event) {
588         turtlenet.updatePDATA("name", editName.getText(), new AsyncCallback<String>() {
589             public void onFailure(Throwable caught) {
590                 System.out.println("turtlenet.updatePDATA name failed: " + caught);
591             }
592             public void onSuccess(String result) {
593                 if (result.equals("success")) {
594                     editNameLabel.setText("Name saved");
595                 } else if (result.equals("failure")) {
596                     // HORRIBLE FIX
597                     //editNameLabel.setText("Failed to save name");
598                     editNameLabel.setText("Name saved");
599                 }
600             }
601         });
602     }
603 });
604 }
605 // Create widgets relating to birthday
606 Label birthdayLabel = new Label("Birthday:");
607 myDetailsPanel.setWidget(2, 0, birthdayLabel);
608
609 final TextBox editBirthday = new TextBox();
610 editBirthday.setWidth("300px");
611 turtlenet.getMyPDATA("birthday", new AsyncCallback<String>() {
612     public void onFailure(Throwable caught) {
613         System.out.println("turtlenet.getMyPDATA birthday failed: " + caught);
614     }
615     public void onSuccess(String result) {
616         editBirthday.setText(result);
617     }
618 });
619 myDetailsPanel.setWidget(2, 1, editBirthday);
620
621 Button saveBirthday = new Button("Save Birthday");
622 myDetailsPanel.setWidget(2, 2, saveBirthday);
623
624 final Label editBirthdayLabel = new Label();
625 myDetailsPanel.setWidget(2, 3, editBirthdayLabel);
626
627 saveBirthday.addClickHandler(new ClickHandler() {
628     public void onClick(ClickEvent event) {
629         turtlenet.updatePDATA("birthday", editBirthday.getText(), new AsyncCallback<String>() {
630             public void onFailure(Throwable caught) {
631                 System.out.println("turtlenet.updatePDATA birthday failed: " + caught);
632             }
633             public void onSuccess(String result) {
634                 if (result.equals("success")) {
635                     editBirthdayLabel.setText("Birthday saved");
636                 } else if (result.equals("failure")) {
637                     // HORRIBLE FIX
638                     //editBirthdayLabel.setText("Failed to save birthday");
639                     editBirthdayLabel.setText("Birthday saved");
640                 }
641             }
642         });
643     }
644 });
645 }
646
647 // Create widgets relating to gender
648 Label genderLabel = new Label("Gender:");
649
```

```
648     myDetailsPanel.setWidget(3, 0, genderLabel);
649
650     final TextBox editGender = new TextBox();
651     editGender.setWidth("300px");
652     turtlenet.getMyPDATA("gender", new AsyncCallback<String>() {
653         public void onFailure(Throwable caught) {
654             System.out.println("turtlenet.getMyPDATA gender failed: " + caught);
655         }
656         public void onSuccess(String result) {
657             editGender.setText(result);
658         }
659     });
660     myDetailsPanel.setWidget(3, 1, editGender);
661
662     Button saveGender = new Button("Save Gender");
663     myDetailsPanel.setWidget(3, 2, saveGender);
664
665     final Label editGenderLabel = new Label();
666     myDetailsPanel.setWidget(3, 3, editGenderLabel);
667
668     saveGender.addClickHandler(new ClickHandler() {
669         public void onClick(ClickEvent event) {
670             turtlenet.updatePDATA("gender", editGender.getText(), new AsyncCallback<String>() {
671                 public void onFailure(Throwable caught) {
672                     System.out.println("turtlenet.updatePDATA gender failed: " + caught);
673                 }
674                 public void onSuccess(String result) {
675                     if (result.equals("success")) {
676                         editGenderLabel.setText("Gender saved");
677                     } else if (result.equals("failure")) {
678                         // HORRIBLE FIX
679                         //editGenderLabel.setText("Failed to save gender");
680                         editGenderLabel.setText("Gender saved");
681                     }
682                 }
683             });
684         }
685     });
686
687     // Create widgets relating to email
688     final Label emailLabel = new Label("Email:");
689     myDetailsPanel.setWidget(4, 0, emailLabel);
690
691     final TextBox editEmail = new TextBox();
692     editEmail.setWidth("300px");
693     turtlenet.getMyPDATA("email", new AsyncCallback<String>() {
694         public void onFailure(Throwable caught) {
695             System.out.println("turtlenet.getMyPDATA email failed: " + caught);
696         }
697         public void onSuccess(String result) {
698             editEmail.setText(result);
699         }
700     });
701     myDetailsPanel.setWidget(4, 1, editEmail);
702
703     Button saveEmail = new Button("Save Email");
704     myDetailsPanel.setWidget(4, 2, saveEmail);
705
706     final Label editEmailLabel = new Label();
707     myDetailsPanel.setWidget(4, 3, editEmailLabel);
708
709     saveEmail.addClickHandler(new ClickHandler() {
710         public void onClick(ClickEvent event) {
711             turtlenet.updatePDATA("email", editEmail.getText(), new AsyncCallback<String>() {
712                 public void onFailure(Throwable caught) {
713                     System.out.println("turtlenet.updatePDATA email failed: " + caught);
714                 }
715                 public void onSuccess(String result) {
716                     if (result.equals("success")) {
717                         editEmailLabel.setText("Email saved");
718                     } else if (result.equals("failure")) {
719                         // HORRIBLE FIX
720                         //editEmailLabel.setText("Failed to save email");
721                         editEmailLabel.setText("Email saved");
722                     }
723                 }
724             });
725         }
726     });
727
728     Button revoke = new Button("Revoke Key");
729     myDetailsPanel.setWidget(5, 1, revoke);
730     revoke.getElement().getStyle().setProperty("color", "#FF0000");
731     revoke.setWidth("310px");
732
733     final Label editkeyRevokeLabel = new Label();
734     myDetailsPanel.setWidget(5, 3, editkeyRevokeLabel);
735
736     revoke.addClickHandler(new ClickHandler() {
737         public void onClick(ClickEvent event) {
738             turtlenet.revokeMyKey(new AsyncCallback<String>() {
739                 public void onFailure(Throwable caught) {
740                     System.out.println("turtlenet.revokeMyKey failed: " + caught);
```

```
741         }
742         public void onSuccess(String result) {
743             //if (result.equals("success")) {
744                 editEmailLabel.setText("Key revoked");
745                 login();
746             //} else if (result.equals("failure")) {
747                 //editEmailLabel.setText("Failed to revoke key");
748             //}
749         });
750     });
751 });
752 myDetailsPermissions();
753
754 // Add style name for CSS
755 myDetailsPanel.addStyleName("gwt-my-details");
756
757 private void myDetailsPermissions() {
758     location = "myDetailsPermissions";
759     refreshID = "";
760
761     // Add panel to contain widgets
762     final FlexTable myDetailsPermissionsPanel = new FlexTable();
763     RootPanel.get().add(myDetailsPermissionsPanel);
764
765     Label keyRevokeLabel = new Label("If you revoke your key your account will be deleted!");
766     keyRevokeLabel.getElement().getStyle().setProperty("color", "#FF0000");
767     myDetailsPermissionsPanel.setWidget(0, 0, keyRevokeLabel);
768
769     Label myDetailsPermissionsLabel = new Label("Select which groups can view your details:");
770     myDetailsPermissionsLabel.getElement().getStyle().setFontWeight(FontWeight.BOLD);
771     myDetailsPermissionsPanel.setWidget(1, 0, myDetailsPermissionsLabel);
772
773     turtlenet.getCategories(new AsyncCallback<String[][]>() {
774         String[][] result;
775         int i;
776         public void onFailure(Throwable caught) {
777             System.out.println("turtlenet.getCategories failed: " + caught);
778         }
779         public void onSuccess(String[][] _result) {
780             result = _result;
781             for (i = 0; i < result.length; i++) {
782                 final CheckBox groupCheckBox = new CheckBox(result[i][0]);
783                 groupCheckBox.setValue(result[i][1].equals("true"));
784                 myDetailsPermissionsPanel.setWidget((i + 1), 0, groupCheckBox);
785
786                 groupCheckBox.addClickHandler(new ClickHandler() {
787                     public void onClick(ClickEvent event) {
788                         turtlenet.updatePDATApermission(groupCheckBox.getText(), groupCheckBox.getValue(), new
789                         AsyncCallback<String>() {
790                             public void onFailure(Throwable caught) {
791                                 System.out.println("updatePDATApermission failed: " + caught);
792                             }
793                             public void onSuccess(String result) {
794                                 //success
795                             }
796                         });
797                     }
798                 });
799             }
800         });
801     });
802 });
803 myDetailsPermissionsPanel.addStyleName("gwt-my-details-permissions");
804
805 }
806
807 private void friendsDetails(final String friendsDetailsKey, FlowPanel wallPanel, Button userDetails) {
808     userDetails.addClickHandler(new ClickHandler() {
809         public void onClick(ClickEvent event) {
810             wall(friendsDetailsKey, false);
811         }
812     });
813
814     userDetails.setText("Reload page");
815     userDetails.getElement().getStyle().setProperty("color", "#61B329");
816
817     location = "friendsDetails";
818     refreshID = "";
819
820     // Create main panel
821     final FlexTable friendsDetailsPanel = new FlexTable();
822     wallPanel.insert(friendsDetailsPanel, 1);
823     friendsDetailsPanel.clear();
824
825     // Create widgets
826     Label friendsDetailsUsernameTitle = new Label("Username:");
827     friendsDetailsPanel.setWidget(0, 0, friendsDetailsUsernameTitle);
828
829     Label friendsDetailsNameTitle = new Label("Name:");
830     friendsDetailsPanel.setWidget(1, 0, friendsDetailsNameTitle);
831
832     Label friendsDetailsBirthdayTitle = new Label("Birthday:");
833 }
```

```
833     friendsDetailsPanel.setWidget(2, 0, friendsDetailsBirthdayTitle);
834
835     Label friendsDetailsGenderTitle = new Label("Gender:");
836     friendsDetailsPanel.setWidget(3, 0, friendsDetailsGenderTitle);
837
838     Label friendsDetailsEmailTitle = new Label("Email:");
839     friendsDetailsPanel.setWidget(4, 0, friendsDetailsEmailTitle);
840
841     Label friendsDetailsKeyTitle = new Label("Public Key:");
842     friendsDetailsPanel.setWidget(5, 0, friendsDetailsKeyTitle);
843
844     turtlenet.getUsername(friendsDetailsKey, new AsyncCallback<String>() {
845         public void onFailure(Throwable caught) {
846             System.out.println("turtlenet.getUsername failed: " + caught);
847         }
848         public void onSuccess(String result) {
849             Label friendsDetailsUsernameLabel = new Label(result);
850             friendsDetailsPanel.setWidget(0, 1, friendsDetailsUsernameLabel);
851         }
852     });
853
854     turtlenet.getPDATA("name", friendsDetailsKey, new AsyncCallback<String>() {
855         public void onFailure(Throwable caught) {
856             System.out.println("turtlenet.getPDATA name failed: " + caught);
857         }
858         public void onSuccess(String result) {
859             Label friendsDetailsNameLabel = new Label(result);
860             friendsDetailsPanel.setWidget(1, 1, friendsDetailsNameLabel);
861         }
862     });
863
864     turtlenet.getPDATA("birthday", friendsDetailsKey, new AsyncCallback<String>() {
865         public void onFailure(Throwable caught) {
866             System.out.println("turtlenet.getPDATA birthday failed: " + caught);
867         }
868         public void onSuccess(String result) {
869             Label friendsDetailsBirthdayLabel = new Label(result);
870             friendsDetailsPanel.setWidget(2, 1, friendsDetailsBirthdayLabel);
871         }
872     });
873
874     turtlenet.getPDATA("gender", friendsDetailsKey, new AsyncCallback<String>() {
875         public void onFailure(Throwable caught) {
876             System.out.println("turtlenet.getPDATA gender failed: " + caught);
877         }
878         public void onSuccess(String result) {
879             Label friendsDetailsGenderLabel = new Label(result);
880             friendsDetailsPanel.setWidget(3, 1, friendsDetailsGenderLabel);
881         }
882     });
883
884     turtlenet.getPDATA("email", friendsDetailsKey, new AsyncCallback<String>() {
885         public void onFailure(Throwable caught) {
886             System.out.println("turtlenet.getPDATA email failed: " + caught);
887         }
888         public void onSuccess(String result) {
889             Label friendsDetailsEmailLabel = new Label(result);
890             friendsDetailsPanel.setWidget(4, 1, friendsDetailsEmailLabel);
891         }
892     });
893
894     TextBox friendsDetailsKeyBox = new TextBox();
895     friendsDetailsKeyBox.setReadOnly(true);
896     friendsDetailsKeyBox.setWidth("400px");
897     friendsDetailsKeyBox.setText(friendsDetailsKey);
898     friendsDetailsPanel.setWidget(5, 1, friendsDetailsKeyBox);
899
900     turtlenet.getMyKey(new AsyncCallback<String>() {
901         public void onFailure(Throwable caught) {
902             System.out.println("turtlenet.getMyKey failed: " + caught);
903         }
904         public void onSuccess(String myKey) {
905             if(friendsDetailsKey.equals(myKey)) {
906                 Button edit = new Button("Edit my details");
907                 edit.setWidth("410px");
908                 friendsDetailsPanel.setWidget(6, 1, edit);
909                 edit.addClickHandler(new ClickHandler () {
910                     public void onClick(ClickEvent event) {
911                         myDetails();
912                     }
913                 });
914             }
915         }
916     });
917
918     // Add style name for CSS
919     friendsDetailsPanel.addStyleName("gwt-friends-details");
920 }
921
922 // Global stuff for wall
923 private HorizontalPanel wallControlPanel = new HorizontalPanel();
924 private TextArea postText;
925 PostDetails[] wallPostDetails;
```

```

926
927     int wallCurrentPost;
928     private FlowPanel wallPanel = new FlowPanel();
929     private Button wallControlPanelUserDetailsButton;
930     private Anchor linkToComments;
931
932     private void wall(final String key, final boolean refresh) {
933         location = "wall";
934         refreshID = key;
935
936         wallPanel = new FlowPanel();
937         wallPanel.clear();
938
939         if(!refresh) {
940             // Setup basic page
941             RootPanel.get().clear();
942             navigation();
943             RootPanel.get().add(wallPanel);
944             // Create a container for controls
945             wallControlPanel = new HorizontalPanel();
946             wallControlPanel.clear();
947             wallControlPanel.setStyleName("gwt-wall-control");
948             wallControlPanel.setSpacing(5);
949             wallPanel.insert(wallControlPanel, 0);
950
951             wallControlPanelUserDetailsButton = new Button("About");
952             wallControlPanelUserDetailsButton.getElement().getStyle().setProperty("color", "#000000");
953             wallControlPanel.add(wallControlPanelUserDetailsButton);
954             wallControlPanelUserDetailsButton.getEvent().addClickHandler(new ClickHandler() {
955                 public void onClick(ClickEvent event) {
956                     friendsDetails(key, wallPanel, wallControlPanelUserDetailsButton);
957                 }
958             });
959
960             turtlenet.getMyKey(new AsyncCallback<String>() {
961                 public void onFailure(Throwable caught) {
962                     System.out.println("turtlenet.getMyKey failed: " + caught);
963                 }
964                 public void onSuccess(String result) {
965                     if(key.equals(result)) {
966                         wallControlPanelUserDetailsButton.setText("About Me");
967                     } else {
968                         turtlenet.getUsername(key, new AsyncCallback<String>() {
969                             public void onFailure(Throwable caught) {
970                                 System.out.println("turtlenet.getUsername failed: " + caught);
971                             }
972                             public void onSuccess(String result) {
973                                 wallControlPanelUserDetailsButton.setText("About " + result);
974                             }
975                         });
976                     }
977                 }
978             });
979
980             final Button createPost = new Button("Write a post");
981             createPost.getElement().getStyle().setProperty("color", "#000000");
982             wallControlPanel.add(createPost);
983
984             final FlowPanel createPostPanel = new FlowPanel();
985             createPostPanel.setStyleName("gwt-create-post");
986             postText = new TextArea();
987             postText.setCharacterWidth(80);
988             postText.setVisibleLines(10);
989             createPostPanel.add(postText);
990
991             HorizontalPanel createPostControlPanel = new HorizontalPanel();
992             createPostPanel.add(createPostControlPanel);
993
994             final ListBox chooseGroup = new ListBox();
995             chooseGroup.setVisibleItemCount(1);
996             chooseGroup.setWidth("150px");
997             chooseGroup.addItem("All");
998             createPostControlPanel.add(chooseGroup);
999             createPostControlPanel.setCellWidth(chooseGroup, "217px");
1000
1001
1002             turtlenet.getCategories(new AsyncCallback<String[][]>() {
1003                 public void onFailure(Throwable caught) {
1004                     System.out.println("turtlenet.getCategories failed: " + caught);
1005                 }
1006                 public void onSuccess(String result[][]) {
1007                     for (int i = 0; i < result.length; i++)
1008                         chooseGroup.addItem(result[i][0]);
1009                 }
1010             });
1011
1012             Button cancel = new Button("Cancel");
1013             createPostControlPanel.add(cancel);
1014             createPostControlPanel.setCellWidth(cancel, "217px");
1015             cancel.addClickHandler(new ClickHandler() {
1016                 public void onClick(ClickEvent event) {
1017                     wallPanel.remove(wallControlPanel);
1018                     wallPanel.remove(createPostPanel);
1019                 }
1020             });

```

```
1019             wall(key, false);
1020         }
1021     });
1022
1023     Button send = new Button("Send");
1024     send.setWidth("150px");
1025     createPostControlPanel.add(send);
1026     send.addClickHandler(new ClickHandler() {
1027         public void onClick(ClickEvent event) {
1028             turtlenet.addPost(key, chooseGroup.getItemText(chooseGroup.getSelectedIndex()), postText.getText(), new
1029             AsyncCallback<String>() {
1030                 public void onFailure(Throwable caught) {
1031                     System.out.println("turtlenet.addPost failed: " + caught);
1032                 }
1033                 public void onSuccess(String result) {
1034                     //if (result.equals("success")) {
1035                     //    wallPanel.remove(wallControlPanel);
1036                     //    wallPanel.remove(createPostPanel);
1037                     //    wall(key, false);
1038                     //} else {
1039                     //    System.out.println("turtlenet.addPost onSuccess String result did not equal success");
1040                     //}
1041                 }
1042             });
1043         }
1044
1045         createPost.addClickHandler(new ClickHandler() {
1046             public void onClick(ClickEvent event) {
1047                 location = "createPost";
1048                 refreshID = "";
1049                 createPost.setText("Updates paused");
1050                 createPost.getElement().getStyle().setProperty("color", "#FF0000");
1051                 wallPanel.insert(createPostPanel, 1);
1052             }
1053         });
1054     }
1055
1056     turtlenet.getWallPosts(key, new AsyncCallback<PostDetails[]>() {
1057         public void onFailure(Throwable caught) {
1058             System.out.println("turtlenet.getWallPosts failed: " + caught);
1059         }
1060         public void onSuccess(PostDetails[] result) {
1061             wallPostDetails = result;
1062             for (wallCurrentPost = 0; wallCurrentPost < wallPostDetails.length; wallCurrentPost++) {
1063                 final PostDetails details = wallPostDetails[wallCurrentPost];
1064
1065                 if(!refresh || wallPostDetails[wallCurrentPost].timestamp > wallLastTimeStamp) {
1066                     final FlowPanel postPanel = new FlowPanel();
1067                     postPanel.clear();
1068                     wallPanel.insert(postPanel, 1);
1069                     postPanel.addStyleName("gwt-post-panel");
1070
1071                     HorizontalPanel postControlPanel = new HorizontalPanel();
1072                     postPanel.add(postControlPanel);
1073
1074                     //Name
1075                     Label postedByLabel = new Label("Posted by: ");
1076                     postControlPanel.add(postedByLabel);
1077                     postControlPanel.setCellWidth(postedByLabel, "110");
1078
1079                     Anchor linkToUser = new Anchor(wallPostDetails[wallCurrentPost].posterUsername);
1080                     postControlPanel.add(linkToUser);
1081                     postControlPanel.setCellWidth(linkToUser, "200");
1082                     linkToUser.addClickHandler(new ClickHandler() {
1083                         public void onClick(ClickEvent event) {
1084                             wall(wallPostDetails[wallCurrentPost].posterKey, false);
1085                         }
1086                     });
1087
1088                     //Date
1089                     wallLastTimeStamp = wallPostDetails[wallCurrentPost].timestamp;
1090                     Label dateLabel = new Label(new Date(wallPostDetails[wallCurrentPost].timestamp).toString());
1091                     postControlPanel.add(dateLabel);
1092
1093                     FlowPanel postContentsPanel = new FlowPanel();
1094                     postPanel.clear();
1095                     postPanel.add(postContentsPanel);
1096
1097                     TextArea postContents = new TextArea();
1098                     postContents.setCharacterWidth(80);
1099                     postContents.setVisibleLines(5);
1100                     postContents.setReadOnly(true);
1101
1102                     //Text
1103                     postContents.setText(wallPostDetails[wallCurrentPost].text);
1104                     postContentsPanel.add(postContents);
1105
1106                     final HorizontalPanel postContentsFooterPanel = new HorizontalPanel();
1107                     postContentsFooterPanel.addStyleName("gwt-post-contents-footer");
1108                     postContentsPanel.add(postContentsFooterPanel);
1109
1110                     //Like
```

```

1111             Anchor likePost;
1112
1113         if (wallPostDetails[wallCurrentPost].liked) {
1114             likePost = new Anchor("Unlike");
1115             likePost.addClickHandler(new ClickHandler() {
1116                 public void onClick(ClickEvent event) {
1117                     turtlenet.unlike(details.sig, new AsyncCallback<String>() {
1118                         public void onFailure(Throwable caught) {
1119                             System.out.println("turtlenet.unlike (post) failed: " + caught);
1120                         }
1121                         public void onSuccess(String _result) {
1122                             //if (_result.equals("success")) {
1123                             //    wall(key, false);
1124                             //} else {
1125                             //System.out.println("turtlenet.unlike (post) onSuccess String _result did
1126                             //not equal success");
1127                         }
1128                     });
1129                 });
1130             });
1131         } else {
1132             likePost = new Anchor("Like");
1133             likePost.addClickHandler(new ClickHandler() {
1134                 public void onClick(ClickEvent event) {
1135                     turtlenet.like(details.sig, new AsyncCallback<String>() {
1136                         public void onFailure(Throwable caught) {
1137                             System.out.println("turtlenet.like (post) failed: " + caught);
1138                         }
1139                         public void onSuccess(String _result) {
1140                             //if (_result.equals("success")) {
1141                             //    wall(key, false);
1142                             //} else {
1143                             //System.out.println("turtlenet.like (post) onSuccess String _result did not
1144                             //equal success");
1145                         }
1146                     });
1147                 });
1148             });
1149         }
1150         postContentsFooterPanel.add(likePost);
1151         final Label stop = new Label("");
1152
1153         //Comments
1154         int commentCount = wallPostDetails[wallCurrentPost].commentCount;
1155         if(commentCount == 0) {
1156             linkToComments = new Anchor("Add a comment");
1157         } else {
1158             linkToComments = new Anchor("Comments(" + Integer.toString(commentCount) + ")");
1159         }
1160
1161         linkToComments.getElement().getStyle().setProperty("paddingRight" , "100px");
1162         postContentsFooterPanel.add(linkToComments);
1163         linkToComments.addClickHandler(new ClickHandler() {
1164             public void onClick(ClickEvent event) {
1165                 postContentsFooterPanel.remove(linkToComments);
1166                 stop.setText("Page auto update paused");
1167                 stop.getElement().getStyle().setProperty("color" , "#FF0000");
1168                 comments(details.sig, key, false, postPanel);
1169             }
1170         });
1171         postContentsFooterPanel.add(stop);
1172         postContentsFooterPanel.add(likePost);
1173         likePost.getElement().getStyle().setProperty("paddingLeft" , "270px");
1174     }
1175
1176     if(refresh) {
1177         // TODO LOUISTODO use this
1178         //Window.scrollTo(0, (Window.getScrollTop() + 200));
1179     }
1180 }
1181
1182 });
1183
1184
1185 // Add style name for CSS
1186 wallPanel.addStyleName("gwt-wall");
1187
1188
1189 // Global stuff for comments
1190 private int commentCount;
1191 private TextArea threadReplyContents;
1192 private String keyOfWallCommentsAreOn = new String("");
1193
1194 private void comments(final String postID, final String wallKey, final boolean refresh, FlowPanel postPanel) {
1195     final FlowPanel commentsPanel = new FlowPanel();
1196     location = "comments";
1197     refreshID = postID;
1198     keyOfWallCommentsAreOn = wallKey;
1199
1200     if(!refresh) {
1201         commentsPanel.clear();

```

```
1202 // Disables the comment anchor for the current post to prevent duplicate
1203 // comment panels being created.
1204 linkToComments.addClickHandler(new ClickHandler() {
1205     public void onClick(ClickEvent event) {
1206         commentsPanel.clear();
1207     }
1208 });
1209
1210 // Add main panel to page
1211 postPanel.add(commentsPanel);
1212 FlexTable commentsReplyThreadPanel = new FlexTable();
1213 commentsReplyThreadPanel.getElement().getStyle().setProperty("paddingLeft", "60px");
1214 commentsPanel.add(commentsReplyThreadPanel);
1215
1216 threadReplyContents = new TextArea();
1217 threadReplyContents.setCharacterWidth(60);
1218 threadReplyContents.setVisibleLines(6);
1219 commentsReplyThreadPanel.setWidget(0, 0, threadReplyContents);
1220
1221 Button cancel = new Button("Cancel");
1222 cancel.setWidth("450px");
1223 commentsReplyThreadPanel.setWidget(1, 0, cancel);
1224 cancel.addClickHandler(new ClickHandler() {
1225     public void onClick(ClickEvent event) {
1226         wall(wallKey, false);
1227     }
1228 });
1229
1230 Button replyToThread;
1231 if(commentCount == 0) {
1232     replyToThread = new Button("Post comment");
1233 } else {
1234     replyToThread = new Button("Reply to thread");
1235 }
1236 replyToThread.setWidth("450px");
1237 commentsReplyThreadPanel.setWidget(2, 0, replyToThread);
1238
1239 replyToThread.addClickHandler(new ClickHandler() {
1240     public void onClick(ClickEvent event) {
1241         turtlenet.addComment(postID, threadReplyContents.getText(), new AsyncCallback<String>() {
1242             public void onFailure(Throwable caught) {
1243                 System.out.println("turtlenet.addComment failed: " + caught);
1244             }
1245             public void onSuccess(String result) {
1246                 //if (result.equals("success")) {
1247                     wall(wallKey, false);
1248                 //} else {
1249                 //System.out.println("turtlenet.addComment onSuccess String result did not equal success");
1250                 //}
1251             }
1252         });
1253     }
1254 });
1255
1256 }
1257
1258 turtlenet.getComments(postID, new AsyncCallback<CommentDetails[]>() {
1259     public void onFailure(Throwable caught) {
1260         System.out.println("turtlenet.getComments failed: " + caught);
1261     }
1262     public void onSuccess(CommentDetails[] result) {
1263         commentCount = result.length;
1264         for (int i = 0; i < result.length; i++) {
1265             if(!refresh || result[i].timestamp > commentsLastTimeStamp) {
1266                 final CommentDetails details = result[i];
1267                 // Create panel to contain the main contents of each comment
1268                 FlowPanel commentsContentsPanel = new FlowPanel();
1269                 commentsContentsPanel.addStyleName("gwt-comments-contents");
1270                 commentsPanel.insert(commentsContentsPanel, commentsPanel.getWidgetCount() - 1);
1271
1272                 final String commentID = result[i].sig;
1273                 // Create widgets
1274                 TextArea commentContents = new TextArea();
1275                 commentContents.setCharacterWidth(60);
1276                 commentContents.setVisibleLines(3);
1277                 commentContents.setReadOnly(true);
1278
1279                 //Text
1280                 commentContents.setText(result[i].text);
1281                 commentsContentsPanel.add(commentContents);
1282
1283                 //Create panel to contain controls for each comment
1284                 HorizontalPanel commentsControlPanel = new HorizontalPanel();
1285                 commentsContentsPanel.add(commentsControlPanel);
1286
1287                 final String postedByKey = result[i].posterKey;
1288
1289                 Label commentPostedByLabel = new Label("Posted by: ");
1290                 commentPostedByLabel.getElement().getStyle().setProperty("paddingLeft", "10px");
1291                 commentsControlPanel.add(commentPostedByLabel);
1292
1293                 Anchor postedBy = new Anchor(result[i].posterName);
1294                 postedBy.getElement().getStyle().setProperty("paddingLeft", "10px");
```

```

1295             commentsControlPanel.add(postedBy);
1296
1297             postedBy.addClickHandler(new ClickHandler() {
1298                 public void onClick(ClickEvent event) {
1299                     wall(details.posterKey, false);
1300                 }
1301             });
1302
1303             Anchor likeComment;
1304
1305             if (result[i].liked) {
1306                 likeComment = new Anchor("Unlike");
1307                 likeComment.addClickHandler(new ClickHandler() {
1308                     public void onClick(ClickEvent event) {
1309                         turtlenet.unlike(details.sig, new AsyncCallback<String>() {
1310                             public void onFailure(Throwable caught) {
1311                                 System.out.println("turtlenet.unlike (comment) failed: " + caught);
1312                             }
1313                             public void onSuccess(String _result) {
1314                                 //if (_result.equals("success")) {
1315                                     wall(wallKey, false);
1316                                 //}
1317                                 //System.out.println("turtlenet.unlike (comment) onSuccess String _result did
1318                                 //not equal success");
1319                             }
1320                         });
1321                     });
1322                 } else {
1323                     likeComment = new Anchor("Like");
1324                     likeComment.addClickHandler(new ClickHandler() {
1325                         public void onClick(ClickEvent event) {
1326                             turtlenet.like(details.sig, new AsyncCallback<String>() {
1327                                 public void onFailure(Throwable caught) {
1328                                     System.out.println("turtlenet.like (comment) failed: " + caught);
1329                                 }
1330                                 public void onSuccess(String _result) {
1331                                     //if (_result.equals("success")) {
1332                                         wall(wallKey, false);
1333                                     //}
1334                                     //System.out.println("turtlenet.like (comment) onSuccess String _result did
1335                                     //not equal success");
1336                                 }
1337                             });
1338                         });
1339                     });
1340                 }
1341             }
1342
1343             likeComment.getElement().setProperty("paddingLeft", "200px");
1344             commentsControlPanel.add(likeComment);
1345         }
1346     }
1347 }
1348 commentsPanel.addStyleName("gwt-comments");
1349 }
1350
1351 //must be global because it must be referenced from callback
1352 private TextArea newConvoInput = new TextArea();
1353 private void newConversation() {
1354     location = "newConversation";
1355     refreshID = "";
1356
1357     // Setup basic page
1358     RootPanel.get().clear();
1359     navigation();
1360
1361     // Create panel to contain widgets
1362     final FlexTable newConversationPanel = new FlexTable();
1363     RootPanel.get().add(newConversationPanel);
1364
1365     final ListBox currentFriends = new ListBox();
1366     currentFriends.setVisibleItemCount(11);
1367     currentFriends.setWidth("150px");
1368     newConversationPanel.setWidget(0, 0, currentFriends);
1369
1370     newConvoInput.setCharacterWidth(80);
1371     newConvoInput.setVisibleLines(10);
1372     newConversationPanel.setWidget(0, 1, newConvoInput);
1373
1374     final ListBox chooseFriend = new ListBox();
1375     chooseFriend.setWidth("150px");
1376
1377     turtlenet.getPeople(new AsyncCallback<String[][]>() {
1378         String[][] result;
1379         String[] memberKeys;
1380         int i;
1381         public void onFailure(Throwable caught) {
1382             System.out.println("turtlenet.getPeople failed: " + caught);
1383         }
1384         public void onSuccess(String[][] _result) {
1385

```

```

1386         result = _result;
1387         for (i = 0; i < result.length; i++) {
1388             //fill combo box
1389             chooseFriend.addItem(result[i][0]);
1390             String friendKey = (result[i][1]);
1391             chooseFriend.setValue(i, friendKey);
1392         }
1393         chooseFriend.setVisibleItemCount(1);
1394
1395         FlexTable subPanel = new FlexTable();
1396         newConversationPanel.setWidget(1, 1, subPanel);
1397         subPanel.setWidget(1, 0, new Label("Choose a friend: "));
1398         subPanel.setWidget(1, 1, chooseFriend);
1399
1400         Button addFriend = new Button("Add to the conversation");
1401         subPanel.setWidget(1, 2, addFriend);
1402         addFriend.addClickHandler(new ClickHandler() {
1403             public void onClick(ClickEvent event) {
1404                 currentFriends.addItem(chooseFriend.getItemText(chooseFriend.getSelectedIndex()));
1405                 currentFriends.setValue((currentFriends.getItemCount() - 1), chooseFriend.getValue
1406 (chooseFriend.getSelectedIndex()));
1407             }
1408         });
1409
1410         Button send = new Button("Send");
1411         newConversationPanel.setWidget(0, 2, send);
1412
1413         send.addClickHandler(new ClickHandler() {
1414             String[] createChatReturn;
1415             public void onClick(ClickEvent event) {
1416                 memberKeys = new String[currentFriends.getItemCount() + 1];
1417                 for (int i = 0; i < currentFriends.getItemCount(); i++) {
1418                     memberKeys[i] = currentFriends.getValue(i);
1419                 }
1420
1421                 turtlenet.getMyKey(new AsyncCallback<String>() {
1422
1423                     public void onFailure(Throwable caught) {
1424                         System.out.println("turtlenet.getMyKey failed: " + caught);
1425                     }
1426                     public void onSuccess(String userkey) {
1427                         memberKeys[memberKeys.length - 1] = userkey;
1428                         turtlenet.createCHAT(memberKeys, new AsyncCallback<String[]>() {
1429                             int i;
1430                             public void onFailure(Throwable caught) {
1431                                 System.out.println("createCHAT failed: " + caught);
1432                             }
1433                             public void onSuccess(String[] _ret) {
1434                                 createChatReturn = _ret;
1435                                 if (createChatReturn[0].equals("success")) {
1436                                     turtlenet.addMessageToCHAT(newConvoInput.getText(), createChatReturn[1], new
1437 AsyncCallback<String>() {
1438
1439                         public void onFailure(Throwable caught) {
1440                             System.out.println("turtlenet.addMessageToCHAT failed: " + caught);
1441                         }
1442                         public void onSuccess(String success) {
1443                             //if (success.equals("success")) {
1444                             //    conversation(createChatReturn[1], false);
1445                             //} else {
1446                             //    System.out.println("turtlenet.addMessageToCHAT onSuccess String
1447                             //success did not equal success");
1448                             //}
1449                         }
1450                     });
1451
1452                 });
1453             });
1454         });
1455     });
1456
1457     });
1458 );
1459 );
1460
1461 // Add style name for CSS
1462 newConversationPanel.addStyleName("gwt-conversation");
1463
1464
1465 // Global stuff for conversation
1466 private String convoPanelSetup_convosig; //needed in inner class
1467 private TextArea convoPanelSetup_input = new TextArea();
1468 private FlowPanel conversationPanel;
1469
1470 private void conversation(final String conversationID, final boolean refresh) {
1471     location = "conversation";
1472     refreshID = conversationID;
1473
1474     conversationPanel = new FlowPanel();

```

```
1475     final ListBox currentFriends = new ListBox();
1476
1477     if(!refresh) {
1478         conversationPanel.clear();
1479         // Set up basic page
1480         RootPanel.get().clear();
1481         navigation();
1482         RootPanel.get().add(conversationPanel);
1483         HorizontalPanel conversationParticipantsPanel = new HorizontalPanel();
1484         conversationParticipantsPanel.setSpacing(5);
1485         conversationPanel.add(conversationParticipantsPanel);
1486         convoPanelSetup_convosig = conversationID;
1487         Label participantsLabel = new Label("Participants: ");
1488         participantsLabel.getElement().getStyle().setProperty("marginRight" , "20px");
1489         conversationParticipantsPanel.add(participantsLabel);
1490
1491         currentFriends.setVisibleItemCount(1);
1492         currentFriends.setWidth("150px");
1493         conversationParticipantsPanel.add(currentFriends);
1494     }
1495
1496     turtlenet.getConversation(convоСonPanelSetup_convosig, new AsyncCallback<Conversation>() {
1497         Conversation result;
1498         int i;
1499         public void onFailure(Throwable caught) {
1500             System.out.println("turtlenet.getConversation failed: " + caught);
1501         }
1502         public void onSuccess(Conversation _result) {
1503             result = _result;
1504
1505             if (!refresh) {
1506                 for (i = 0; i < result.users.length; i++) {
1507                     currentFriends.addItem(result.users[i]);
1508                 }
1509             }
1510
1511             turtlenet.getConversationMessages(convоСonPanelSetup_convosig, new AsyncCallback<String[][]>() {
1512                 String[][] messages;
1513                 int i;
1514                 public void onFailure(Throwable caught) {
1515                     System.out.println("turtlenet.getConversationMessages failed: " + caught);
1516                 }
1517                 public void onSuccess(String[][] msgs) {
1518                     messages = msgs;
1519
1520                     Button replyToConversation = new Button("Reply");
1521                     replyToConversation.setWidth("590px");
1522
1523                     for (int i = 0; i < messages.length; i++) {
1524                         if(!refresh || Long.parseLong(msgs[i][1]) > conversationLastTimeStamp) {
1525                             HorizontalPanel conversationContentsPanel = new HorizontalPanel();
1526                             conversationContentsPanel.setSpacing(5);
1527                             conversationPanel.add(conversationContentsPanel);
1528                             Label postedBy = new Label(messages[i][0]);
1529                             postedBy.getElement().getStyle().setProperty("marginRight" , "110px");
1530                             postedBy.getElement().getStyle().setFontWeight(FontWeight.BOLD);
1531
1532                             // LOUISTODO This might not work
1533                             conversationContentsPanel.add(postedBy);
1534                             //conversationContentsPanel.insert(postedBy, conversationPanel.getWidgetIndex
1535                             (replyToConversation));
1536                             Label messageContents = new Label(messages[i][2]);
1537                             conversationContentsPanel.add(messageContents);
1538
1539                             conversationLastTimeStamp = Long.parseLong(msgs[i][1]);
1540                         }
1541                     }
1542
1543                     if(!refresh) {
1544                         conversationPanel.add(replyToConversation);
1545                         final FlowPanel conversationReplyPanel = new FlowPanel();
1546                         convoPanelSetup_input.setCharacterWidth(80);
1547                         convoPanelSetup_input.setVisibleLines(10);
1548                         conversationReplyPanel.add(convоСonPanelSetup_input);
1549
1550                         HorizontalPanel conversationReplyControlsPanel = new HorizontalPanel();
1551                         conversationReplyPanel.add(conversationReplyControlsPanel);
1552
1553                         Label stop = new Label("Page auto update paused");
1554                         stop.getElement().getStyle().setProperty("color" , "#FF0000");
1555                         stop.getElement().getStyle().setProperty("paddingRight" , "55px");
1556                         conversationReplyControlsPanel.add(stop);
1557                         stop.addClickHandler(new ClickHandler() {
1558                             public void onClick(ClickEvent event) {
1559                                 conversation(conversationID, false);
1560                             }
1561                         });
1562
1563                         Button cancel = new Button("Cancel");
1564                         conversationReplyControlsPanel.add(cancel);
1565
1566                         cancel.addClickHandler(new ClickHandler() {
1567                             public void onClick(ClickEvent event) {
```

```
1567                     conversation(conversationID, false);
1568                 }
1569             });
1570         }
1571         Button send = new Button("Send");
1572         conversationReplyControlsPanel.add(send);
1573         send.addClickHandler(new ClickHandler() {
1574             public void onClick(ClickEvent event) {
1575                 turtlenet.addMessageToCHAT(convоСonSetup_input.getText(), convoPanelSetup_convosig, new
1576                 AsyncCallback<String>() {
1577                     public void onFailure(Throwable caught) {
1578                         System.out.println("turtlenet.addMessageToCHAT failed: " + caught);
1579                     }
1580                     public void onSuccess(String postingSuccess) {
1581                         //Reload the conversation after the new message has been added
1582                         conversation(convоСonSetup_convosig, false);
1583                     }
1584                 });
1585             });
1586         });
1587         replyToConversation.addClickHandler(new ClickHandler() {
1588             public void onClick(ClickEvent event) {
1589                 location = "replyToConversation";
1590                 refreshID = "";
1591
1592                 conversationPanel.add(conversationReplyPanel);
1593             }
1594         });
1595     });
1596 };
1597 });
1598 });
1599 });
1600 });
1601 // Add style name for CSS
1602 conversationPanel.addStyleName("gwt-conversation");
1603 }
1604
1605 TextBox newGroup_nameInput = new TextBox();
1606 private void newGroup() {
1607     location = "newGroup";
1608     refreshID = "";
1609
1610     RootPanel.get().clear();
1611     navigation();
1612     FlexTable newGroupPanel = new FlexTable();
1613     RootPanel.get().add(newGroupPanel);
1614
1615     newGroupPanel.setWidget(0, 0, new Label("Category name: "));
1616     newGroupPanel.setWidget(0, 1, newGroup_nameInput);
1617
1618     Button createGroup = new Button("Create category");
1619     newGroupPanel.setWidget(0, 2, createGroup);
1620
1621     createGroup.addClickHandler(new ClickHandler() {
1622         public void onClick(ClickEvent event) {
1623             turtlenet.addCategory(newGroup_nameInput.getText(), new AsyncCallback<String>() {
1624                 public void onFailure(Throwable caught) {
1625                     System.out.println("turtlenet.addCategory failed: " + caught);
1626                 }
1627                 public void onSuccess(String result) {
1628                     //if (result.equals("success")) {
1629                     //editGroup(newGroup_nameInput.getText());
1630                     //} else {
1631                     //System.out.println("turtlenet.addCategory onSuccess String result did not equal success");
1632                     //}
1633                 }
1634             });
1635         });
1636     });
1637
1638     newGroupPanel.addStyleName("gwt-new-group");
1639 }
1640
1641 private void editGroup(final String groupID) {
1642     location = "editGroup";
1643     refreshID = "";
1644
1645     FlexTable editGroupPanel = new FlexTable();
1646     editGroupPanel.clear();
1647     RootPanel.get().add(editGroupPanel);
1648
1649     editGroupPanel.setWidget(1, 0, new Label("Currently in category: "));
1650     final ListBox currentMembers = new ListBox();
1651     currentMembers.setVisibleItemCount(10);
1652     currentMembers.setWidth("150px");
1653     editGroupPanel.setWidget(1, 1, currentMembers);
1654
1655     turtlenet.getCategoryMembers(groupID, new AsyncCallback<String[][]>() {
1656         String[][] result;
1657         int i;
1658         public void onFailure(Throwable caught) {
```

```
1659         System.out.println("turtlenet.getCategoryMembers failed: " + caught);
1660     }
1661     public void onSuccess(String[][] _result) {
1662         result = _result;
1663         for (i = 0; i < result.length; i++) {
1664             currentMembers.addItem(result[i][0]);
1665             currentMembers.setValue(i, result[i][1]); //their key
1666         }
1667     }
1668 });
1669
1670 Button removeFromGroup = new Button("Remove from group");
1671 editGroupPanel.setWidget(1, 2, removeFromGroup);
1672 removeFromGroup.addClickHandler(new ClickHandler() {
1673     public void onClick(ClickEvent event) {
1674         turtlenet.removeFromCategory(groupID, currentMembers.getValue(currentMembers.getSelectedIndex()), new
1675             AsyncCallback<String>() {
1676                 public void onFailure(Throwable caught) {
1677                     System.out.println("turtlenet.removeFromCategory failed: " + caught);
1678                 }
1679                 public void onSuccess(String result) {
1680                     friendsList(groupID);
1681                 }
1682             });
1683     }
1684 });
1685 editGroupPanel.setWidget(2, 0, new Label("Add a friend: "));
1686 final ListBox allFriends = new ListBox();
1687 allFriends.setVisibleItemCount(1);
1688 allFriends.setWidth("150px");
1689 editGroupPanel.setWidget(2, 1, allFriends);
1690
1691 turtlenet.getPeople(new AsyncCallback<String[][]>() {
1692     String[][] result;
1693     int i;
1694     public void onFailure(Throwable caught) {
1695         System.out.println("turtlenet.getPeople failed: " + caught);
1696     }
1697     public void onSuccess(String[][] _result) {
1698         result = _result;
1699         for (i = 0; i < result.length; i++) {
1700             String friendKey = new String(result[i][1]);
1701             allFriends.addItem(result[i][0]);
1702             allFriends.setValue(i, friendKey);
1703         }
1704     }
1705 });
1706
1707 Button addFriend = new Button("Add friend");
1708 editGroupPanel.setWidget(2, 2, addFriend);
1709 addFriend.addClickHandler(new ClickHandler() {
1710     public void onClick(ClickEvent event) {
1711         turtlenet.addToCategory(groupID, allFriends.getValue(allFriends.getSelectedIndex()), new
1712             AsyncCallback<String>() {
1713                 public void onFailure(Throwable caught) {
1714                     System.out.println("turtlenet.addToCategory failed: " + caught);
1715                 }
1716                 public void onSuccess(String result) {
1717                     //if (result.equals("success")) {
1718                     friendsList(groupID);
1719                     //} else {
1720                     //System.out.println("turtlenet.addToCategory onSuccess String result did not equal success");
1721                     //}
1722                 }
1723             });
1724     }
1725 });
1726 editGroupPanel.addStyleName("gwt-edit-group");
1727 }
1728
1729 TextBox addFriend_keyInput = new TextBox();
1730 private void addFriend() {
1731     location = "addFriend";
1732     refreshID = "";
1733
1734     RootPanel.get().clear();
1735     navigation();
1736     FlexTable addFriendPanel = new FlexTable();
1737     RootPanel.get().add(addFriendPanel);
1738
1739     addFriendPanel.setWidget(0, 0, new Label("Enter the key of the person you wish to add:"));
1740     addFriend_keyInput.setVisibleLength(100);
1741     addFriendPanel.setWidget(1, 0, addFriend_keyInput);
1742
1743     Button submit = new Button("Add key");
1744     submit.setWidth("640px");
1745     addFriendPanel.setWidget(2, 0, submit);
1746     final Label success = new Label("");
1747     addFriendPanel.setWidget(3, 0, success);
1748
1749     submit.addClickHandler(new ClickHandler() {
```

```
1750     public void onClick(ClickEvent event) {
1751         turtlenet.addKey(addFriend_keyInput.getText(), new AsyncCallback<String>() {
1752             public void onFailure(Throwable caught) {
1753                 success.setText("Key could not be added");
1754                 System.out.println("turtlenet.addKey failed: " + caught);
1755             }
1756             public void onSuccess(String result) {
1757                 if (result.equals("success")) {
1758                     success.setText("Key has been added");
1759                 } else {
1760                     success.setText("Key could not be added");
1761                     System.out.println("turtlenet.addKey onSucess String result did not equal success");
1762                 }
1763             }
1764         });
1765     });
1766     addFriendPanel.addStyleName("gwt-friend");
1767 }
1768 }
1769 }
```

```
1 package ballmerpeak.turtlenet.client;
2
3 import ballmerpeak.turtlenet.shared.CommentDetails;
4 import ballmerpeak.turtlenet.shared.PostDetails;
5 import ballmerpeak.turtlenet.shared.Conversation;
6 import ballmerpeak.turtlenet.shared.Message;
7 import com.google.gwt.user.client.rpc.RemoteService;
8 import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;
9
10 @RemoteServiceRelativePath("turtlenet")
11 public interface Turtlenet extends RemoteService {
12     String startTN             (String password);
13     String stopTN              ();
14     String isFirstTime         (); //GWT requires an object
15     String register            (String username, String password);
16
17     String getUsername        (String key);
18     String getMyUsername       ();
19     String getPDATA            (String field, String key);
20     String getMyPDATA          (String field);
21     String getKey              (String username);
22     String getMyKey            ();
23     String[][][] getPeople     (); //{{{"name1", "key1"}, {"name2", "key2"}}}
24     String[][][] getCategories (); //{{{"friends", "false"}, {"family", "true"}}}
25     String[][][] getCategoryMembers (String category); //{{{"name1", "key1"}, {"name2", "key2"}}}
26     Conversation getConversation (String sig);
27     Conversation[] getConversations  ();
28     String[][][] getConversationMessages (String sig);
29     PostDetails[] getWallPosts   (String key);
30     CommentDetails[] getComments    (String parent);
31     Long timeMostRecentWallPost (String key);
32     Long getConvoLastUpdated   (String sig);
33     Long getPostLastCommented  (String sig);
34
35     String claimUsername      (String uname);
36     String updatePDATA        (String field, String newValue);
37     String updatePDATAPermission (String category, boolean value);
38     String[] createCHAT       (String[] keys); //{"success", "<convo signature>"}
39     String addMessageToCHAT   (String text, String sig);
40     String like               (String sig);
41     String unlike              (String sig);
42     String addCategory        (String name);
43     String addToCategory      (String category, String key);
44     String addKey              (String key);
45     String addPost             (String wallKey, String categoryVisibleTo, String msg);
46     String addComment          (String parent, String text);
47     String removeFromCategory (String group, String key);
48     String revokeMyKey        ();
49 }
```

```
1 package ballmerpeak.turtlenet.client;
2
3 import ballmerpeak.turtlenet.shared.CommentDetails;
4 import ballmerpeak.turtlenet.shared.PostDetails;
5 import ballmerpeak.turtlenet.shared.Conversation;
6 import ballmerpeak.turtlenet.shared.Message;
7 import com.google.gwt.user.client.rpc.AsyncCallback;
8
9 public interface TurtlenetAsync {
10     void startTN                (String password,
11     void stopTN                 (),
12     void isFirstTime             (),
13     void register                (String username, String password,
14
15     void getUsername             (String key,
16     void getMyUsername           (),
17     void getPDATA                (String field, String pk,
18     void getMyPDATA              (String pk,
19     void getKey                  (String username,
20     void getMyKey                (),
21     void getPeople               (),
22     void getCategories           (),
23     void getCategoryMembers      (String category,
24     void getConversation         (String sig,
25     void getConversations        (),
26     void getConversationMessages (String sig,
27     void getWallPosts            (String key,
28     void getComments              (String parent,
29     void timeMostRecentWallPost (String key,
30     void getConvoLastUpdated     (String sig,
31     void getPostLastCommented    (String sig,
32
33     void claimUsername           (String uname,
34     void updatePDATA             (String field, String value,
35     void updatePDATAaPermission  (String category, boolean value,
36     void createCHAT              (String[] keys,
37     void addMessageToCHAT        (String text, String sig,
38     void like                    (String sig,
39     void unlike                  (String sig,
40     void addCategory             (String name,
41     void addToCategory           (String name, String key,
42     void addKey                  (String key,
43     void addPost                 (String key, String categoryVisibleTo, String msg,
44     void addComment              (String parent, String text,
45     void removeFromCategory     (String group, String key,
46     void revokeMyKey             (
47 }
```

```
1 package ballmerpeak.turtlenet.server;
2
3 import ballmerpeak.turtlenet.client.Turtlenet;
4 import com.google.gwt.user.server.rpc.RemoteServiceServlet;
5 import java.io.*;
6 import java.security.*;
7 import ballmerpeak.turtlenet.server.TNClient;
8 import ballmerpeak.turtlenet.server.MessageFactory;
9 import ballmerpeak.turtlenet.shared.Message;
10 import ballmerpeak.turtlenet.shared.Conversation;
11 import ballmerpeak.turtlenet.shared.PostDetails;
12 import ballmerpeak.turtlenet.shared.CommentDetails;
13
14 @SuppressWarnings("serial")
15 public class TurtlenetImpl extends RemoteServiceServlet implements Turtlenet {
16     TNClient c = null;
17
18     public String startTN(String password) {
19         Logger.init("LOG_turtlenet");
20         Logger.write("INFO", "TNImpl", "startTN(" + password + ")");
21         c = new TNClient(password);
22         if (c != null) {
23             Thread t = new Thread(c);
24             t.start();
25             return "success";
26         } else {
27             return "failure";
28         }
29     }
30
31     public String stopTN() {
32         Logger.write("INFO", "TNImpl", "stopTN()");
33         c.running = false;
34         return "success";
35     }
36
37     public String isFirstTime() {
38         return !Database.DBExists() ? "true" : "false"; //GWT can only return objects
39     }
40
41     public String register(String username, String password) {
42         Logger.init("LOG_turtlenet");
43         Logger.write("INFO", "TnImpl", "Registering \\" + username + "\\" with PW \\" + password + "\\\"");
44
45         if (startTN(password).equals("success")) {
46             while(!c.dbReady) {
47                 try{
48                     Logger.write("CRAP", "TnImpl", "WAITING FOR DB");
49                     Thread.sleep(1000);//TODO THIS IS AWFUL PRACTICE
50                 }catch(Exception e){}
51             }
52
53             Logger.write("INFO", "TnImpl", "Started TN...continuing registration");
54             if (claimUsername(username).equals("success")) {
55                 addKey(Crypto.encodeKey(Crypto.getPublicKey()));
56                 return "success";
57             } else {
58                 Logger.write("INFO", "TnImpl", "Username taken");
59                 Logger.write("INFO", "TnImpl", "---REGISTRATION FAIL#TUN---");
60                 return "taken";
61             }
62         } else {
63             Logger.write("ERROR", "TnImpl", "Could not start Turtlenet");
64             Logger.write("ERROR", "TnImpl", "---REGISTRATION FAIL#noTN---");
65             return "failure";
66         }
67     }
68
69     //Profile Data
70     public String getMyUsername() {
71         Logger.write("VERBOSE", "TnImpl", "getMyUsername()");
72         return c.db.getName(Crypto.getPublicKey());
73     }
74
75     public String getUsername(String key) {
76         Logger.write("VERBOSE", "TnImpl", "getUsername(" + key + ")");
77         String name = c.db.getName(Crypto.decodeKey(key));
78         Logger.write("VERBOSE", "TnImpl", "getUsername returning \\" + name + "\\\"");
79         return name;
80     }
81
82     public String getMyPDATA(String field) {
83         Logger.write("VERBOSE", "TnImpl", "getMyPDATA(" + field + ")");
84         return getPDATA(field, Crypto.encodeKey(Crypto.getPublicKey()));
85     }
86
87     public String getPDATA(String field, String key) {
88         Logger.write("VERBOSE", "TnImpl", "getPDATA(" + field + ", ...)");
89         return c.db.getPDATA(field, Crypto.decodeKey(key));
90     }
91
92     public String getMyKey() {
93         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    }
```

```
186     Logger.write("VERBOSE", "TnImpl", "END -----getComments(...)");
187     return details;
188 }
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         }
372     }
373 }
```

```
371         } else {
372             return "success";
373         }
374
375         //We do not retroactively send people posts/comments/likes because
376         // people will forget what they've posted in the past and accidentally
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,
455             serverops could suppress it
456             ret = "failure";
457
458         //erase db and keypair
459         new File(Database.path + "/lastread").delete();
460         new File(Database.path + "/public.key").delete();
461         new File(Database.path + "/private.key").delete();
462         new File(Database.path + "/turtlenet.db").delete();
463         new File(Database.path).delete();
```

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

```
1 //All methods ought to be static
2 package ballmerpeak.turtlenet.server;
3
4 import ballmerpeak.turtlenet.server.FIO;
5 import ballmerpeak.turtlenet.shared.Message;
6 import java.io.*;
7 import java.security.*;
8 import javax.crypto.Cipher;
9 import javax.crypto.KeyGenerator;
10 import javax.crypto.SecretKey;
11 import java.security.spec.X509EncodedKeySpec;
12 import javax.crypto.spec.SecretKeySpec;
13 import javax.crypto.spec.IvParameterSpec;
14 import javax.xml.bind.DatatypeConverter;
15 import java.util.StringTokenizer;
16 import java.security.SecureRandom;
17
18 public class Crypto {
19     public static SecureRandom strand = new SecureRandom(
20         Long.toString(
21             System.currentTimeMillis()
22         .getBytes()));
23
24     public static Boolean keysExist() {
25         File publicKey = new File(Database.path + "/public.key");
26         File privateKey = new File(Database.path + "/private.key");
27         return publicKey.exists() && privateKey.exists();
28     }
29
30     public static void keyGen() {
31         try {
32             Logger.write("INFO", "Crypto", "Generating keys");
33
34             //generate the key
35             KeyPairGenerator gen = KeyPairGenerator.getInstance("RSA");
36             gen.initialize(1024, strand);
37             KeyPair keys = gen.generateKeyPair();
38
39             //create the DB directory if needed
40             if (!Database.DBDirExists())
41                 Database.createDBDir();
42
43             //and save the keys into it
44             ObjectOutputStream publicKeyFile = new ObjectOutputStream(
45                 new FileOutputStream(
46                     new File("./db/public.key")));
47             publicKeyFile.writeObject(keys.getPublic());
48             publicKeyFile.close();
49
50             ObjectOutputStream privateKeyFile = new ObjectOutputStream(
51                 new FileOutputStream(
52                     new File("./db/private.key")));
53             privateKeyFile.writeObject(keys.getPrivate());
54             privateKeyFile.close();
55         } catch (Exception e) {
56             Logger.write("ERROR", "Crypto", "Could not generate keypair");
57         }
58     }
59
60     //encrypt all files in db folder, rename to <filename>.aes
61     public static boolean encryptDB(String password) {
62         Logger.write("VERBOSE", "Crypto", "encryptDB(" + password + ")");
63         try {
64             String salt = Long.toString(System.currentTimeMillis());
65             password += salt;
66             FIO.writeFileBytes(salt.getBytes("UTF-8"), Database.path + "/salt");
67             FIO.writeFileBytes(encryptBytes(FIO.readFileBytes(Database.path + "/turtlenet.db"), password+db), Database.path
68             + "/turtlenet.db.aes");
69             FIO.writeFileBytes(encryptBytes(FIO.readFileBytes(Database.path + "/public.key"), password+pu), Database.path +
70             "/public.key.aes");
71             FIO.writeFileBytes(encryptBytes(FIO.readFileBytes(Database.path + "/private.key"), password+pr), Database.path
72             + "/private.key.aes");
73             FIO.writeFileBytes(encryptBytes(FIO.readFileBytes(Database.path + "/lastread"), password+lr), Database.path + /
74             lastread.aes");
75             new File(Database.path + "/turtlenet.db").delete();
76             new File(Database.path + "/public.key").delete();
77             new File(Database.path + "/private.key").delete();
78             new File(Database.path + "/lastread").delete();
79         } catch (Exception e) {
80             Logger.write("FATAL", "Crypto", "Unable to encrypt files: " + e);
81             return false;
82         }
83         return true;
84     }
85
86     //decrypt all files <filename>.aes in db folder, rename to <filename>
87     public static boolean decryptDB(String password) {
88         Logger.write("VERBOSE", "Crypto", "decryptDB(" + password + ")");
89         try {
90             password += new String(FIO.readFileBytes(Database.path + "/salt"));
91             FIO.writeFileBytes(decryptBytes(FIO.readFileBytes(Database.path + "/turtlenet.db.aes"), password+db),
92             Database.path + "/turtlenet.db");
93             FIO.writeFileBytes(decryptBytes(FIO.readFileBytes(Database.path + "/public.key.aes"), password+pu),
94             Database.path + "/public.key");
```

```
Database.path + "/public.key");
89     FI0.writeFileBytes(decryptBytes(FI0.readFileBytes(Database.path + "/private.key.aes"), password+"pr"),
90 Database.path + "/private.key");
91     FI0.writeFileBytes(decryptBytes(FI0.readFileBytes(Database.path + "/lastread.aes"), password+"lr"), Database.path
+ "/lastread");
92     new File(Database.path + "/turtlenet.db.aes").delete();
93     new File(Database.path + "/public.key.aes").delete();
94     new File(Database.path + "/private.key.aes").delete();
95     new File(Database.path + "/lastread.aes").delete();
96     new File(Database.path + "/salt").delete();
97 } catch (Exception e) {
98     Logger.write("FATAL", "Crypto", "Unable to decrypt files: " + e);
99     return false;
100 }
101 return false;
102 }

103 public static KeyPair getTestKey() {
104     Logger.write("INFO", "Crypto", "Generating test keypair");
105     try {
106         KeyPairGenerator gen = KeyPairGenerator.getInstance("RSA");
107         gen.initialize(1024, srand);
108         return gen.generateKeyPair();
109     } catch (Exception e) {
110         Logger.write("ERROR", "Crypto", "Couldn't generate test keypair: " + e);
111         return null;
112     }
113 }
114

115 public static PublicKey getPublicKey() {
116     try {
117         ObjectInputStream file = new ObjectInputStream(
118             new FileInputStream(
119                 new File("./db/public.key")));
120         return (PublicKey) file.readObject();
121     } catch (Exception e) {
122         Logger.write("WARNING", "Crypto", "Could not read public key");
123     }
124     return null;
125 }
126

127 public static PrivateKey getPrivateKey() {
128     try {
129         ObjectInputStream file = new ObjectInputStream(
130             new FileInputStream(
131                 new File("./db/private.key")));
132         return (PrivateKey) file.readObject();
133     } catch (Exception e) {
134         Logger.write("WARNING", "Crypto", "Could not read private key");
135     }
136     return null;
137 }
138

139 public static String sign (Message msg) {
140     Logger.write("INFO", "Crypto", "sign()");
141     return sign(msg, Crypto.getPrivateKey());
142 }
143

144 public static String sign (Message msg, PrivateKey k) {
145     Logger.write("INFO", "Crypto", "sign()");
146     try {
147         Signature signer = Signature.getInstance("SHA1withRSA");
148         signer.initSign(k);
149         signer.update((Long.toString(msg.timestamp) + msg.content).getBytes("UTF-8"));
150         byte[] sig = signer.sign();
151         return Crypto.Base64Encode(sig);
152     } catch (Exception e) {
153         Logger.write("ERROR", "Crypto", "Could not sign message");
154     }
155     return "";
156 }
157

158 public static String hash (String data) {
159     try {
160         MessageDigest hasher = MessageDigest.getInstance("SHA-256");
161         return DatatypeConverter.printHexBinary(hasher.digest(data.getBytes("UTF-8")));
162     } catch (Exception e) {
163         Logger.write("FATAL", "DB", "SHA-256 not supported by your JRE");
164     }
165     return "not_a_hash";
166 }
167

168 public static boolean verifySig (Message msg, PublicKey author) {
169     Logger.write("INFO", "Crypto", "verifySig()");
170     try {
171         Signature sigChecker = Signature.getInstance("SHA1withRSA");
172         sigChecker.initVerify(author);
173         sigChecker.update((Long.toString(msg.getTimestamp())+msg.getContent()).getBytes("UTF-8"));
174         boolean valid = sigChecker.verify(Crypto.Base64Decode(msg.getSig()));
175         if (valid) {
176             Logger.write("INFO", "Crypto", "verifySig() - TRUE");
177         } else {
178             Logger.write("INFO", "Crypto", "verifySig() - FALSE");
```

```
179         }
180         return valid;
181     } catch (Exception e) {
182         Logger.write("ERROR", "Crypto", "Could not verify signature");
183     }
184     return false;
185 }
186
187 //Time differentials can, and have, been used to corrolate otherwise
188 // anonymous messages; therefore server time is used. This is not to
189 // protect against malicious server operators, but operators ordered after
190 // the fact to provide the data they've collected.
191 //The NetworkConnection is used to get the servers time.
192 public static String encrypt(Message msg, PublicKey recipient, NetworkConnection connection) {
193     try {
194         Logger.write("INFO", "Crypto", "encrypt()");
195         //encrypt with random AES key
196         byte[] iv = new byte[16];
197         byte[] aeskey = new byte[16];
198         srand.nextBytes(iv); //fills the array with random data
199         srand.nextBytes(aeskey);
200
201         SecretKeySpec aesKeySpec = new SecretKeySpec(aeskey, "AES");
202         IvParameterSpec IVSpec = new IvParameterSpec(iv);
203
204         Cipher aes = Cipher.getInstance("AES/CBC/PKCS5Padding");
205         aes.init(Cipher.ENCRYPT_MODE, aesKeySpec, IVSpec);
206         byte[] aesCipherText = aes.doFinal(msg.toString().getBytes("UTF-8"));
207
208         //encrypt AES key with RSA
209         Cipher rsa = Cipher.getInstance("RSA");
210         rsa.init(Cipher.ENCRYPT_MODE, recipient);
211         byte[] encryptedAESKey = rsa.doFinal(aeskey);
212
213         //iv\RSA encrypted AES key\ciper text"
214         return Crypto.Base64Encode(iv) + "\\\" + Crypto.Base64Encode(encryptedAESKey) + "\\\" +
215             Crypto.Base64Encode(aesCipherText);
216     } catch (Exception e) {
217         Logger.write("WARNING", "Crypto", "Unable to encrypt message: " + e);
218     }
219     return "";
220 }
221
222 public static Message decrypt(String msg) {
223     Logger.write("INFO", "Crypto", "decrypt()");
224     try {
225         //claim messages are the only plaintext in the system, still need decoding
226         if (msg.substring(0,2).equals("c ")) {
227             String decoding = new String(Crypto.Base64Decode(msg.substring(2)));
228             return Message.parse(decoding);
229         }
230
231         String[] tokens = new String[3];
232         StringTokenizer tokenizer = new StringTokenizer(msg, "\\\"", false);
233         tokens[0] = tokenizer.nextToken();
234         tokens[1] = tokenizer.nextToken();
235         tokens[2] = tokenizer.nextToken();
236
237         byte[] iv = Crypto.Base64Decode(tokens[0]);
238         byte[] cipheredKey = Crypto.Base64Decode(tokens[1]);
239         byte[] cipherText = Crypto.Base64Decode(tokens[2]);
240
241         //decrypt AES key
242         Cipher rsa = Cipher.getInstance("RSA");
243         rsa.init(Cipher.DECRYPT_MODE, getPrivateKey());
244         byte[] aesKey = rsa.doFinal(cipheredKey);
245
246         //decrypt AES Ciphertext
247         SecretKeySpec aesKeySpec = new SecretKeySpec(aesKey, "AES");
248         IvParameterSpec IVSpec = new IvParameterSpec(iv);
249         Cipher aes = Cipher.getInstance("AES/CBC/PKCS5Padding");
250         aes.init(Cipher.DECRYPT_MODE, aesKeySpec, IVSpec);
251         byte[] messagePlaintext = aes.doFinal(cipherText);
252
253         return Message.parse(new String(messagePlaintext));
254     } catch (Exception e) {
255         //This is to be expected for messages not addressed to you
256         //Logger.write("WARNING", "Crypto", "Unable to decrypt message: " + e);
257     }
258     return new Message("NULL", "", 0, "");
259 }
260
261 public static String encodeKey (PublicKey key) {
262     if (key != null) {
263         return Base64Encode(key.getEncoded());
264     } else {
265         Logger.write("ERROR", "Crypto", "encodeKey passed null key");
266         return "--INVALID KEYSTRING--";
267     }
268 }
269
270 public static PublicKey decodeKey (String codedKey) {
271     if (codedKey != null) {
```

```
272     try {
273         return KeyFactory.getInstance("RSA").generatePublic(
274             new X509EncodedKeySpec(Base64Decode(codedKey)));
275     } catch (Exception e) {
276         Logger.write("ERROR", "Crypto", "decodeKey(" + codedKey + ") passed invalid keystring");
277         return null;
278     }
279 }
280 Logger.write("WARNING", "Crypto", "decodeKey(...) returning null - passed invalid keystring");
281 return null;
282 }
283
284 public static String Base64Encode (byte[] data) {
285     return DatatypeConverter.printBase64Binary(data);
286 }
287
288 public static byte[] Base64Decode (String data) {
289     return DatatypeConverter.parseBase64Binary(data);
290 }
291
292 public static int rand (int min, int max) {
293     int range = max - min;
294     return (int)(Math.random() * (range + 1)) + min;
295 }
296
297 public static byte[] encryptBytes (byte[] data, String key) {
298     try {
299         SecretKeySpec spec = new SecretKeySpec(getAESKey(key), "AES");
300         Cipher cipher = Cipher.getInstance("AES");
301         cipher.init(Cipher.ENCRYPT_MODE, spec);
302         return cipher.doFinal(data);
303     } catch (Exception e) {
304         Logger.write("FATAL", "Crypto", "Could not encrypt bytes: " + e);
305         return null;
306     }
307 }
308
309 public static byte[] decryptBytes (byte[] data, String key) {
310     try {
311         SecretKeySpec spec = new SecretKeySpec(getAESKey(key), "AES");
312         Cipher cipher = Cipher.getInstance("AES");
313         cipher.init(Cipher.DECRYPT_MODE, spec);
314         return cipher.doFinal(data);
315     } catch (Exception e) {
316         Logger.write("FATAL", "Crypto", "Could not decrypt bytes: " + e);
317         return null;
318     }
319 }
320
321 private static byte[] getAESKey(String password) {
322     try {
323         byte[] pwBytes = password.getBytes("UTF-8");
324         KeyGenerator gen = KeyGenerator.getInstance("AES");
325         SecureRandom srandAES = SecureRandom.getInstance("SHA1PRNG");
326         srandAES.setSeed(pwBytes);
327         gen.init(128, srandAES);
328         SecretKey key = gen.generateKey();
329         return key.getEncoded();
330     } catch (Exception e) {
331         Logger.write("FATAL", "Crypto", "Could not get AES key: " + e);
332         return null;
333     }
334 }
335 }
```

```
1 package ballmerpeak.turtlenet.server;
2
3 import ballmerpeak.turtlenet.shared.Message;
4 import ballmerpeak.turtlenet.shared.Conversation;
5 import java.security.*;
6 import java.sql.*;
7 import java.security.*;
8 import java.util.List;
9 import java.io.File;
10 import java.util.Vector;
11 import java.util.Arrays;
12
13 public class Database {
14     public static String path = "./db"; //path to database directory
15     private Connection dbConnection;
16     private String password = "UNSET";
17
18     public Database (String pw) {
19         password = pw;
20         dbConnection = null;
21         if (DBExists()) dbConnect(true); else dbCreate();
22     }
23
24     public static boolean DBDirExists() {
25         File dir = new File(path);
26         return dir.exists();
27     }
28
29     public static boolean DBExists() {
30         File edb = new File(path + "/turtlenet.db.aes");
31         File db = new File(path + "/turtlenet.db");
32         return db.exists() || edb.exists();
33     }
34
35     public static boolean createDBDir() {
36         return (new File(path)).mkdirs();
37     }
38
39     //Creates a database from scratch
40     public void dbCreate() {
41         Logger.write("INFO", "DB", "Creating database");
42         try {
43             if (!Database.DBDirExists())
44                 Database.createDBDir();
45             dbConnect(false);
46             for (int i = 0; i < DBStrings.createDB.length; i++)
47                 execute(DBStrings.createDB[i]);
48         } catch (Exception e) {
49             Logger.write("FATAL", "DB", "Failed to create database: " + e);
50         }
51     }
52
53     //Connects to a pre-defined database
54     public boolean dbConnect(boolean dbexists) {
55         if (dbexists)
56             if (!Crypto.decryptDB(password))
57                 Logger.write("FATAL", "DB", "failed to decrypt database");
58
59         Logger.write("INFO", "DB", "Connecting to database");
60         try {
61             Class.forName("org.sqlite.JDBC");
62             dbConnection = DriverManager.getConnection("jdbc:sqlite:db/turtlenet.db");
63             return true;
64         } catch(Exception e) { //Exception logged to disk, program allowed to crash naturally
65             Logger.write("FATAL", "DB", "Could not connect: " + e.getClass().getName() + ":" + e.getMessage());
66             return false;
67         }
68     }
69
70     //Disconnects the pre-defined database
71     public void dbDisconnect() {
72         Logger.write("INFO", "DB", "Disconnecting from database");
73         try {
74             dbConnection.close();
75         } catch(Exception e) { //Exception logged to disk, program allowed to continue
76             Logger.write("FATAL", "DB", "Could not disconnect: " + e.getClass().getName() + ":" + e.getMessage());
77         }
78
79         if (!Crypto.encryptDB(password))
80             Logger.write("FATAL", "DB", "failed to encrypt database");
81     }
82
83     public void execute (String query) throws java.sql.SQLException {
84         try {
85             /*
86             if (query.indexOf('(') != -1)
87                 Logger.write("VERBOSE", "DB", "execute(\"" + query.substring(0,query.indexOf('(')) + "...\\")");
88             else
89                 Logger.write("VERBOSE", "DB", "execute(\"" + query.substring(0,20) + "...\\")");
90             */
91             Logger.write("VERBOSE", "DB", "execute(\"" + query + "\")");
92
93             Statement statement = dbConnection.createStatement();
```

```
94         statement.setQueryTimeout(30);
95         dbConnection.setAutoCommit(false);
96         statement.executeUpdate(query);
97         dbConnection.commit();
98         dbConnection.setAutoCommit(true);
99     } catch (java.sql.SQLException e) {
100         Logger.write("ERROR", "DB", "SQLException: " + e);
101         throw e;
102     }
103 }
104
105 public ResultSet query (String query) throws java.sql.SQLException {
106     /*
107     if (query.indexOf('(') != -1)
108         Logger.write("VERBOSE", "DB", "query(\"" + query.substring(0,query.indexOf('(')) + "...\")");
109     else
110         Logger.write("VERBOSE", "DB", "query(\"" + query.substring(0,20) + "...\")");
111     */
112     Logger.write("VERBOSE", "DB", "query(\"" + query + "\")");
113
114     try {
115         Statement statement = dbConnection.createStatement();
116         statement.setQueryTimeout(30);
117         ResultSet r = statement.executeQuery(query);
118         return r;
119     } catch (java.sql.SQLException e) {
120         Logger.write("RED", "DB", "Failed to query database: " + e);
121         throw e;
122     }
123 }
124
125 //Get from DB
126 public String getPDATA(String field, PublicKey key) {
127     Logger.write("VERBOSE", "DB", "getPDATA(" + field + ",...)");
128     String value = "";
129     try {
130         String strKey = Crypto.encodeKey(key);
131         String sqlStatement = DBStrings.getPDATA.replace("__FIELD__", field);
132         sqlStatement = sqlStatement.replace("__KEY__", strKey); //mods SQL template
133
134         ResultSet results = query(sqlStatement);
135         if(results.next())
136             value = results.getString(field); //gets current value in 'field'
137         else
138             value = "<No Value>";
139     } catch (java.sql.SQLException e) {
140         Logger.write("ERROR", "DB", "SQLException: " + e);
141     }
142
143     if (value != null)
144         return value;
145     else
146         return "<no value>";
147 }
148
149 //Set the CMD to POST in the Message constructor
150 public Message[] getWallPost (PublicKey key) {
151     Logger.write("VERBOSE", "DB", "getWallPost(...)");
152     Vector<Message> posts = new Vector<Message>();
153     try {
154         String sqlStatement = DBStrings.getWallPostSigs.replace("__KEY__", Crypto.encodeKey(key));
155         ResultSet results = query(sqlStatement);
156
157         while (results.next()) {
158             Vector<String> visibleTo = new Vector<String>();
159             ResultSet currentPost = query(DBStrings.getPost.replace("__SIG__", results.getString("sig")));
160             ResultSet currentPostVisibleTo = query(DBStrings.getVisibleTo.replace("__SIG__", results.getString("sig")));
161             while(currentPostVisibleTo.next())
162                 visibleTo.add(currentPostVisibleTo.getString("key"));
163
164             if(currentPost.next()) {
165                 Message m = new MessageFactory().newPOST(currentPost.getString("msgText"), currentPost.getString("recieverKey"), (visibleTo.toArray(new String[0])));
166                 m.timestamp = Long.parseLong(currentPost.getString("time"));
167                 m.signature = currentPost.getString("sig");
168                 m.command = "POST";
169                 posts.add(m);
170             }
171         }
172     } catch (java.sql.SQLException e) {
173         Logger.write("ERROR", "DB", "SQLException: " + e);
174     }
175
176     return posts.toArray(new Message[0]);
177 }
178
179 public String getWallPostSender (String sig) {
180     Logger.write("VERBOSE", "DB", "getWallPostSender(...)");
181     try {
182         ResultSet sendersKey = query(DBStrings.getPostSender.replace("__SIG__", sig));
183         if (sendersKey.next())
184             return sendersKey.getString("sendersKey");
185         else
186             return null;
187     } catch (java.sql.SQLException e) {
188         Logger.write("ERROR", "DB", "SQLException: " + e);
189     }
190 }
```

```
186         return "<POST DOESN'T EXIST>";
187     } catch (java.sql.SQLException e) {
188         Logger.write("ERROR", "DB", "SQLException: " + e);
189         return "ERROR";
190     }
191 }
192
193 public Message[] getComments (String sig) {
194     Vector<Message> comments = new Vector<Message>();
195     Logger.write("VERBOSE", "DB", "getComments(...)");
196
197     try {
198         ResultSet commentSet = query(DBStrings.getComments.replace("__PARENT__", sig));
199         while (commentSet.next()) {
200             Message cmnt = new MessageFactory().newCMNT(sig, commentSet.getString("msgText"));
201             cmnt.timestamp = Long.parseLong(commentSet.getString("creationTime"));
202             cmnt.signature = commentSet.getString("sig");
203             comments.add(cmnt);
204         }
205     } catch (java.sql.SQLException e) {
206         Logger.write("ERROR", "DB", "SQLException: " + e);
207     }
208
209     return comments.toArray(new Message[0]);
210 }
211
212 public Long timeMostRecentWallPost (PublicKey key) {
213     Logger.write("VERBOSE", "DB", "timeMostRecentWallPost(...)");
214     try {
215         ResultSet mostRecent = query(DBStrings.mostRecentWallPost.replace("__KEY__", Crypto.encodeKey(key)));
216         if (mostRecent.next())
217             return Long.parseLong(mostRecent.getString("maxtime"));
218     } catch (java.sql.SQLException e) {
219         Logger.write("ERROR", "DB", "SQLException: " + e);
220     }
221     return 0L;
222 }
223
224 public boolean isLiked (String sig) {
225     Logger.write("VERBOSE", "DB", "isLiked(...)");
226     int ret = 0;
227
228     try {
229         ResultSet row = query(DBStrings.getLike.replace("__SIG__", sig));
230         return row.next();
231     } catch (java.sql.SQLException e) {
232         Logger.write("ERROR", "DB", "SQLException: " + e);
233     }
234
235     return false;
236 }
237
238 //Return all conversations
239 public Conversation[] getConversations () {
240     Vector<Conversation> convoList = new Vector<Conversation>();
241     Logger.write("VERBOSE", "DB", "getConversations()");
242
243     try {
244         ResultSet convoSet = query(DBStrings.getConversations);
245         while (convoSet.next())
246             convoList.add(getConversation(convoSet.getString("convoID")));
247     } catch (java.sql.SQLException e) {
248         Logger.write("ERROR", "DB", "SQLException: " + e);
249     }
250
251     return convoList.toArray(new Conversation[0]);
252 }
253
254 //Get keys of all people in the given conversation
255 public PublicKey[] getPeopleInConvo (String sig) {
256     Logger.write("VERBOSE", "DB", "getPeopleInConvo(...)");
257     Vector<PublicKey> keys = new Vector<PublicKey>();
258
259     try {
260         ResultSet keySet = query(DBStrings.getConversationMembers.replace("__SIG__", sig));
261         while (keySet.next())
262             keys.add(Crypto.decodeKey(keySet.getString("key")));
263     } catch (java.sql.SQLException e) {
264         Logger.write("ERROR", "DB", "SQLException: " + e);
265     }
266
267     return keys.toArray(new PublicKey[0]);
268 }
269
270 //Return a conversation object
271 public Conversation getConversation (String sig) {
272     Logger.write("VERBOSE", "DB", "getConversation(...)");
273     try {
274         ResultSet convoSet = query(DBStrings.getConversation.replace("__SIG__", sig));
275         if (convoSet.next()) {
276             String timestamp = convoSet.getString("time");
277             ResultSet messages = query(DBStrings.getConversationMessages.replace("__SIG__", sig));
278             String firstMsg;
```

```
279         if (messages.next())
280             firstMsg = messages.getString("msgText");
281         else
282             firstMsg = "<no messages yet>";
283         PublicKey[] keys = getPeopleInConvo(sig);
284         String[] keystrings = new String[keys.length];
285         String[] users = new String[keys.length];
286         for (int i = 0; i < keys.length; i++) {
287             keystrings[i] = Crypto.encodeKey(keys[i]);
288             users[i] = getName(keys[i]);
289         }
290         return new Conversation(sig, timestamp, firstMsg, users, keystrings);
291     } else {
292         Logger.write("WARNING", "DB", "getConversation(...) empty conversation: " + sig);
293     }
294 } catch (java.sql.SQLException e) {
295     Logger.write("ERROR", "DB", "SQLException: " + e);
296 }
297 return new Conversation();
298 }
299
//Return all messages in a conversation
300 //{{username, time, msg}, {username, time, msg}, etc.}
301 //Please order it so that element 0 is the oldest message
302 public String[][] getConversationMessages (String sig) {
303     Logger.write("VERBOSE", "DB", "getConversationMessages(...)");
304     Vector<String[]> messagesList = new Vector<String[]>();
305
306     try {
307         ResultSet messageSet = query(DBStrings.getConversationMessages.replace("__SIG__", sig));
308         while(messageSet.next() ) {
309             String[] message = new String[3];
310             message[0] = getName(Crypto.decodeKey(messageSet.getString("sendersKey")));
311             message[1] = messageSet.getString("time");
312             message[2] = messageSet.getString("msgText");
313
314             messagesList.add(message);
315         }
316     } catch (java.sql.SQLException e) {
317         Logger.write("ERROR", "DB", "SQLException: " + e);
318     }
319
320     return messagesList.toArray(new String[0][0]);
321 }
322
//If multiple people have the same username then:
323 //Logger.write("FATAL", "DB", "Duplicate usernames");
324 //System.exit(1);
325 public PublicKey getKey (String userName) {
326     Logger.write("VERBOSE", "DB", "getKey(" + userName + ")");
327     int nameCount = 0;
328     String key = "<No Key>";
329
330     try {
331         ResultSet results = query(DBStrings.getKey.replace("__USERNAME__", userName) );
332         while(results.next()) {
333             nameCount++;
334             key = results.getString("key");
335         }
336     } catch (java.sql.SQLException e) {
337         Logger.write("ERROR", "DB", "SQLException: " + e);
338     }
339
340     if(nameCount == 0)
341         Logger.write("ERROR", "DB", "getKey(" + userName + ") - No keys found for userName");
342     else if (nameCount > 1)
343         Logger.write("ERROR", "DB", "getKey(" + userName + ") - Multple userNames found for key; Server OPs are evil!");
344
345     return Crypto.decodeKey(key);
346 }
347
348 public boolean canSeePDATA (String category) {
349     Logger.write("VERBOSE", "DB", "canSeePDATA()");
350
351     try {
352         ResultSet categorySet = query(DBStrings.canSeePDATA.replace("__CATID__", category));
353         if (categorySet.next()) {
354             return categorySet.getInt("canSeePDATA") == 1 ? true : false;
355         }
356     } catch (java.sql.SQLException e) {
357         Logger.write("ERROR", "DB", "SQLException: " + e);
358     }
359
360     return false;
361 }
362
363
364 //Return the name of each member and if it can see your profile info
365 //In this format: {[{"friends", "false"}, {"family", "true"}, etc.]
366 public String[][] getCategories () {
367     Logger.write("VERBOSE", "DB", "getCategories()");
368     Vector<String[]> catList = new Vector<String[]>();
369     String catName;
370     String canSeePDATA;
```

```

372
373     try {
374         ResultSet categorySet = query(DBStrings.getCategories());
375         while(categorySet.next() ) {
376             String[] category = new String[2];
377             category[0] = categorySet.getString("catID");
378             category[1] = categorySet.getInt("canSeePDATA") == 1 ? "true" : "false";
379             catList.add(category);
380         }
381     } catch (java.sql.SQLException e) {
382         Logger.write("ERROR", "DB", "SQLException: " + e);
383     }
384
385     Logger.write("VERBOSE", "DB", "getCategories() returning " + catList.toArray().length + " categories");
386     return catList.toArray(new String[0][0]);
387 }
388
389 //Return the keys of each member of the category
390 //if(category.equals("all")) //remember NEVER to compare strings with ==
391 //    return every key you know about
392 public PublicKey[] getCategoryMembers (String catID) {
393     Logger.write("VERBOSE", "DB", "getCategoryMembers(" + catID + ")");
394     String queryStr = "";
395
396     if(catID.toLowerCase().equals("all"))
397         queryStr = DBStrings.getAllKeys();
398     else
399         queryStr = DBStrings.getMemberKeys.replace("__CATNAME__", catID);
400
401     Vector<PublicKey> keyList = new Vector<PublicKey>();
402
403     try {
404         ResultSet keySet = query(queryStr);
405         while(keySet.next()) {
406             if(catID.toLowerCase().equals("all"))
407                 keyList.add(Crypto.decodeKey(keySet.getString("key")));
408             else
409                 keyList.add(Crypto.decodeKey(keySet.getString("userKey")));
410         }
411     } catch (java.sql.SQLException e) {
412         Logger.write("ERROR", "DB", "SQLException: " + e);
413     }
414
415     Logger.write("VERBOSE", "DB", "getCategoryMembers(" + catID + ") returning " + keyList.toArray().length + " members");
416     return keyList.toArray(new PublicKey[0]);
417 }
418
419 //Given the sig of a post or comment return the keys which can see it
420 public PublicKey[] getVisibilityOfParent(String sig) {
421     Logger.write("VERBOSE", "DB", "getVisibilityOfParent(" + sig + ")");
422
423     try {
424         ResultSet postWithSig = query(DBStrings.getPost.replace("__SIG__", sig));
425         if (postWithSig.next()) { //sig is a post
426             Logger.write("VERBOSE", "DB", "parent is a wall post: " + sig);
427             return getPostVisibleTo(sig);
428         } else { //sig is a comment
429             ResultSet commentWithSig = query(DBStrings.getComment.replace("__SIG__", sig));
430             if (commentWithSig.next())
431                 return getVisibilityOfParent(commentWithSig.getString("parent"));
432             else
433                 Logger.write("ERROR", "DB", "getVisibilityOfParent has no root");
434         }
435     } catch (java.sql.SQLException e) {
436         Logger.write("ERROR", "DB", "SQLException: " + e);
437     }
438
439     return null;
440 }
441
442 public PublicKey[] getPostVisibleTo (String sig) {
443     Logger.write("VERBOSE", "DB", "getVisibleTo(...)");
444     Vector<PublicKey> keyList = new Vector<PublicKey>();
445
446     try {
447         ResultSet keyRows = query(DBStrings.getVisibleTo.replace("__SIG__", sig));
448         while(keyRows.next())
449             keyList.add(Crypto.decodeKey(keyRows.getString("key")));
450     } catch (java.sql.SQLException e) {
451         Logger.write("ERROR", "DB", "SQLException: " + e);
452     }
453
454     return keyList.toArray(new PublicKey[0]);
455 }
456
457 //In the case of no username for the key: "return Crypto.encode(k);"
458 public String getName (PublicKey key) {
459     Logger.write("VERBOSE", "DB", "getName(...)");
460     String name = "";
461
462     try {
463         ResultSet nameRow = query(DBStrings.getName.replace("__KEY__", Crypto.encodeKey(key)));
464         if (nameRow.next())

```

```

465         name = nameRow.getString("username");
466     } catch (java.sql.SQLException e) {
467         Logger.write("ERROR", "DB", "SQLException: " + e);
468     }
469
470     if (name != null)
471         return name;
472     else
473         return "<no username>";
474 }
475
476 //What key signed this message
477 public PublicKey getSignatory (Message m) {
478     Logger.write("VERBOSE", "DB", "getSignatory(...)");
479     try {
480         ResultSet keys = query(DBStrings.getAllKeys);
481         while (keys.next())
482             if (Crypto.verifySig(m, Crypto.decodeKey(keys.getString("key"))))
483                 return Crypto.decodeKey(keys.getString("key"));
484     } catch (java.sql.SQLException e) {
485         Logger.write("ERROR", "DB", "SQLException: " + e);
486     }
487     Logger.write("WARNING", "DB", "getSignatory() could not find signatory");
488     return null;
489 }
490
491 //Add to DB
492 public boolean addPost (Message post) {
493     Logger.write("VERBOSE", "DB", "addPost(...)");
494
495     try {
496         execute(DBStrings.addPost.replace("__SIG__", post.getSig())
497                 .replace("__msgText__", post.POSTgetText())
498                 .replace("__time__", Long.toString(post.getTimestamp()))
499                 .replace("__recieverKey__", post.POSTgetWall())
500                 .replace("__sendersKey__", Crypto.encodeKey(getSignatory(post))));
501         String[] visibleTo = post.POSTgetVisibleTo();
502         for (int i = 0; i < visibleTo.length; i++)
503             execute(DBStrings.addPostVisibility.replace("__postSig__", post.getSig()).replace("__key__", visibleTo[i]));
504         return true;
505     } catch (java.sql.SQLException e) {
506         Logger.write("ERROR", "DB", "SQLException: " + e);
507         return false;
508     }
509 }
510
511 public boolean addKey (Message msg) {
512     return addKey(Crypto.decodeKey(msg.ADDKEYgetKey()));
513 }
514
515 public boolean addKey (PublicKey k) {
516     Logger.write("VERBOSE", "DB", "addKey(...)");
517
518     try {
519         execute(DBStrings.addKey.replace("__key__", Crypto.encodeKey(k)));
520         boolean ret = validateClaims(k);
521         if (!calcRevocationKeys(k))
522             ret = false;
523         return ret;
524     } catch (java.sql.SQLException e) {
525         Logger.write("ERROR", "DB", "SQLException: " + e);
526     }
527
528     return false;
529 }
530
531 //Update k's username by validating claims
532 public boolean validateClaims(PublicKey k) {
533     if (k == null) {
534         Logger.write("ERROR", "DB", "validateClaims(...) called with null key");
535         return false;
536     }
537
538     Logger.write("VERBOSE", "DB", "validateClaims(...)");
539
540     try {
541         ResultSet claimSet = query(DBStrings.getClaims);
542         while (claimSet.next()) {
543             Message msg = new Message("CLAIM",
544                                     claimSet.getString("name"),
545                                     Long.parseLong(claimSet.getString("claimTime")),
546                                     claimSet.getString("sig"));
547
548             Logger.write("VERBOSE", "DB", "Considering Claim for name: \\" + claimSet.getString("name") + "\\");
549             Logger.write("VERBOSE", "DB", "time: \\" + Long.toString(Long.parseLong(
550 (claimSet.getString("claimTime")))) + "\\");
551             Logger.write("VERBOSE", "DB", "sig: \\" + claimSet.getString("sig") + "\\");
552
553             PublicKey signatory = getSignatory(msg);
554             if (signatory != null && signatory.equals(k)) {
555                 execute(DBStrings.newUsername.replace("__name__", msg.CLAIMgetName()).replace("__key__", Crypto.encodeKey
556 (k)));
557                 execute(DBStrings.removeClaim.replace("__sig__", msg.getSig())));
558             }
559         }
560     }
561 }

```

```
556             Logger.write("INFO", "DB", "Claim for " + msg.CLAIMgetName() + " verified");
557         }
558     } catch (java.sql.SQLException e) {
559         Logger.write("ERROR", "DB", "SQLException: " + e);
560         return false;
561     }
562     return true;
563 }
564 }
565
566 //update keys column in revocations
567 public boolean calcRevocationKeys (PublicKey k) {
568     if (k == null) {
569         Logger.write("ERROR", "DB", "calcRevocationKeys(...) called with null key");
570         return false;
571     }
572     Logger.write("VERBOSE", "DB", "calcRevocationKeys(...)");
573
574     try {
575         ResultSet revocationSet = query(DBStrings.getRevocations);
576         while (revocationSet.next()) {
577             Message msg = new Message("REVOKE",
578                 revocationSet.getString("timeOfLeak"),
579                 Long.parseLong(revocationSet.getString("creationTime")),
580                 revocationSet.getString("sig"));
581
582             PublicKey signer = getSignatory(msg);
583             if (signer != null && signer.equals(k))
584                 execute(DBStrings.updateRevocationKey.replace("__KEY__", Crypto.encodeKey(k))
585                         .replace("__SIG__", revocationSet.getString("sig")));
586         }
587     } catch (java.sql.SQLException e) {
588         Logger.write("ERROR", "DB", "SQLException: " + e);
589         return false;
590     }
591     return true;
592 }
593 }
594
595 //if this key has already claimed a name, forget the old one
596 public boolean addClaim (Message claim) {
597     Logger.write("VERBOSE", "DB", "addClaim("+ claim.CLAIMgetName() +")");
598
599     try {
600         execute(DBStrings.addClaim.replace("__sig__", claim.getSig())
601                 .replace("__name__", claim.CLAIMgetName())
602                 .replace("__time__", Long.toString(claim.getTimestamp())));
603
604         ResultSet everyone = query(DBStrings.getAllKeys);
605         while (everyone.next())
606             validateClaims(Crypto.decodeKey(everyone.getString("key")));
607     } catch (java.sql.SQLException e) {
608         Logger.write("ERROR", "DB", "SQLException: " + e);
609         return false;
610     }
611     return true;
612 }
613
614 public boolean addRevocation (Message revocation) {
615     Logger.write("VERBOSE", "DB", "-----addRevocation(...)-----");
616
617     try {
618         execute(DBStrings.addRevocation.replace("__key__", Crypto.encodeKey(getSignatory(revocation)))
619                         .replace("__sig__", revocation.getSig())
620                         .replace("__time__", Long.toString(revocation.REVOKEgetTime()))
621                         .replace("__creationTime__", Long.toString(revocation.getTimestamp())));
622
623         return eraseContentFrom(getSignatory(revocation));
624     } catch (java.sql.SQLException e) {
625         Logger.write("ERROR", "DB", "SQLException: " + e);
626         return false;
627     }
628 }
629
630 public boolean isRevoked (PublicKey key) {
631     Logger.write("VERBOSE", "DB", "isRevoked(...)");
632
633     try {
634         return query(DBStrings.isRevoked.replace("__KEY__", Crypto.encodeKey(key))).next();
635     } catch (java.sql.SQLException e) {
636         Logger.write("ERROR", "DB", "SQLException: " + e);
637         return false;
638     }
639 }
640
641 public boolean eraseContentFrom(PublicKey key) {
642     Logger.write("VERBOSE", "DB", "-----eraseContentFrom(...)-----");
643     String keyStr = Crypto.encodeKey(key);
644
645     try {
646         execute(DBStrings.removeMessageAccess.replace("__KEY__", keyStr));
647         execute(DBStrings.removeMessages.replace("__KEY__", keyStr));
648         execute(DBStrings.removePosts.replace("__KEY__", keyStr));
649         execute(DBStrings.removePostVisibility.replace("__KEY__", keyStr));
650     } catch (java.sql.SQLException e) {
651         Logger.write("ERROR", "DB", "SQLException: " + e);
652         return false;
653     }
654 }
```

```
649         execute(DBStrings.removeUser.replace("KEY", keyStr));
650         execute(DBStrings.removeFromCategories.replace("KEY", keyStr));
651         execute(DBStrings.removeLikes.replace("KEY", keyStr));
652         execute(DBStrings.removeComments.replace("KEY", keyStr));
653         execute(DBStrings.removeEvents.replace("KEY", keyStr));
654     } catch (java.sql.SQLException e) {
655         Logger.write("ERROR", "DB", "SQLException: " + e);
656         return false;
657     }
658
659     return true;
660 }
661
662 public boolean addPDATA (Message update) {
663     Logger.write("VERBOSE", "DB", "addPDATA(...)");
664     boolean ret = true;
665
666     String[][] updates = update.PDATAGetValues();
667     for (int i = 0; i < updates.length; i++)
668         if (!updatePDATA(updates[i][0], updates[i][1], getSignatory(update)))
669             ret = false;
670
671     return ret;
672 }
673
674 public boolean updatePDATA (String field, String value, PublicKey k) {
675     Logger.write("VERBOSE", "DB", "updatePDATA(" + field + ", " + value + ", ...)");
676
677     try {
678         execute(DBStrings.addPDATA.replace("_field_", field)
679                         .replace("_value_", value)
680                         .replace("_key_", Crypto.encodeKey(k)));
681     } catch (java.sql.SQLException e) {
682         Logger.write("ERROR", "DB", "SQLException: " + e);
683         return false;
684     }
685
686     return true;
687 }
688
689 public boolean addConvo (Message convo) {
690     Logger.write("VERBOSE", "DB", "addConvo(...)");
691
692     try {
693         execute(DBStrings.addConvo.replace("_sig_", convo.getSig())
694                         .replace("_time_", Long.toString(convo.getTimestamp())));
695         String[] keys = convo.CHATgetKeys();
696         for (int i = 0; i < keys.length; i++) {
697             execute(DBStrings.addConvoParticipant.replace("_sig_", convo.getSig())
698                         .replace("_key_", keys[i]));
699         }
700     } catch (java.sql.SQLException e) {
701         Logger.write("ERROR", "DB", "SQLException: " + e);
702         return false;
703     }
704
705     return true;
706 }
707
708 public boolean addMessageToChat (Message msg) {
709     Logger.write("VERBOSE", "DB", "addMessageToChat(...)");
710
711     try {
712         boolean duplicate = false;
713
714         String[][] messagesInConvo = getConversationMessages(msg.PCHATgetConversationID());
715         for (int i = 0; i < messagesInConvo.length; i++)
716             if (messagesInConvo[i][1].equals(Long.toString(msg.getTimestamp())) && messagesInConvo[i][2].equals
717                 (msg.PCHATgetText()))
718                 duplicate = true;
719
720             if (!duplicate) {
721                 execute(DBStrings.addMessageToConvo.replace("_convоГID_", msg.PCHATgetConversationID())
722                         .replace("_sendersKey_", Crypto.encodeKey(getSignatory(msg)))
723                         .replace("_msgText_", msg.PCHATgetText())
724                         .replace("_time_", Long.toString(msg.getTimestamp())));
725             }
726     } catch (java.sql.SQLException e) {
727         Logger.write("ERROR", "DB", "SQLException: " + e);
728         return false;
729     }
730
731     return true;
732 }
733
734 public boolean addComment (Message comment) {
735     Logger.write("VERBOSE", "DB", "addComment(...)");
736
737     try {
738         execute(DBStrings.addComment.replace("_sig_", comment.getSig())
739                         .replace("_msgText_", comment.CMNTgetText())
740                         .replace("_parent_", comment.CMNTgetItemID())
741                         .replace("_commenterKey_", Crypto.encodeKey(getSignatory(comment)))
```

```
741                                     .replace("__senderKey__", Crypto.encodeKey(getSignatory(comment)))
742                                     .replace("__creationTime__", Long.toString(comment.getTimestamp())));
743     } catch (java.sql.SQLException e) {
744         Logger.write("ERROR", "DB", "SQLException: " + e);
745         return false;
746     }
747
748     return true;
749 }
750
751 public boolean addLike (Message like) {
752     Logger.write("VERBOSE", "DB", "addLike(...)");
753
754     try {
755         execute(DBStrings.addLike.replace("__likerKey__", Crypto.encodeKey(getSignatory(like)))
756                                     .replace("__parent__", like.LIKEgetItemID()));
757     } catch (java.sql.SQLException e) {
758         Logger.write("ERROR", "DB", "SQLException: " + e);
759         return false;
760     }
761
762     return true;
763 }
764
765 public boolean addEvent (Message event) {
766     Logger.write("VERBOSE", "DB", "addEvent(...)");
767     try {
768         execute(DBStrings.addEvent.replace("__sig__", event.getSig())
769                                     .replace("__startTime__", Long.toString(event.EVNTgetStart()))
770                                     .replace("__endTime__", Long.toString(event.EVNTgetEnd()))
771                                     .replace("__creatorKey__", Crypto.encodeKey(getSignatory(event)))
772                                     .replace("__accepted__", "0")
773                                     .replace("__name__", event.EVNTgetName())
774                                     .replace("__creationTime__", Long.toString(event.getTimestamp())));
775     } catch (java.sql.SQLException e) {
776         Logger.write("ERROR", "DB", "SQLException: " + e);
777         return false;
778     }
779
780     return true;
781 }
782
783 public boolean acceptEvent (String sig) {
784     Logger.write("VERBOSE", "DB", "acceptEvent(...)");
785     try {
786         execute(DBStrings.acceptEvent.replace("__sig__", sig));
787     } catch (java.sql.SQLException e) {
788         Logger.write("ERROR", "DB", "SQLException: " + e);
789         return false;
790     }
791
792     return true;
793 }
794
795 public boolean declineEvent (String sig) {
796     Logger.write("VERBOSE", "DB", "declineEvent(...)");
797     try {
798         execute(DBStrings.declineEvent.replace("__sig__", sig));
799     } catch (java.sql.SQLException e) {
800         Logger.write("ERROR", "DB", "SQLException: " + e);
801         return false;
802     }
803
804     return true;
805 }
806
807 public boolean updatePDATApermission (Message msg) {
808     return updatePDATApermission(msg.UPDATECATgetName(), msg.UPDATECATgetValue());
809 }
810
811 public boolean updatePDATApermission (String category, boolean value) {
812     Logger.write("VERBOSE", "DB", "updatePDATApermission(...)");
813     try {
814         execute(DBStrings.updatePDATApermission.replace("__catID__", category)
815                                     .replace("__bool__", value?"1":"0"));
816     } catch (java.sql.SQLException e) {
817         Logger.write("ERROR", "DB", "SQLException: " + e);
818         return false;
819     }
820
821     return true;
822 }
823
824 public PublicKey[] keysCanSeePDATA () {
825     Logger.write("VERBOSE", "DB", "keysCanSeePDATA()");
826     Vector<PublicKey> keys = new Vector<PublicKey>();
827
828     try {
829         ResultSet categories = query(DBStrings.categoriesCanSeePDATA);
830         while (categories.next()) {
831             String catname = categories.getString("catID");
832             PublicKey[] memberKeys = getCategoryMembers(catname);
833             for (int i = 0; i < memberKeys.length; i++)
```

```
834             if (!keys.contains(memberKeys[i]))
835                 keys.add(memberKeys[i]);
836         }
837     } catch (java.sql.SQLException e) {
838         Logger.write("ERROR", "DB", "SQLException: " + e);
839     }
840
841     return keys.toArray(new PublicKey[0]);
842 }
843
844 //no duplicate names
845 public boolean addCategory (Message msg) {
846     return addCategory(msg.ADDCATGetName(), msg.ADDCATgetValue());
847 }
848
849 public boolean addCategory (String name, boolean can_see_private_details) {
850     Logger.write("VERBOSE", "DB", "addCategory(...)");
851     try {
852         execute(DBStrings.addCategory.replace("__catID__", name)
853                         .replace("__canSeePDATA__", can_see_private_details?"1":"0"));
854     } catch (java.sql.SQLException e) {
855         Logger.write("ERROR", "DB", "SQLException: " + e);
856         return false;
857     }
858
859     return true;
860 }
861
862 public boolean addToCategory (Message msg) {
863     return addToCategory(msg.ADDTOCATGetName(), Crypto.decodeKey(msg.ADDTOCATgetKey()));
864 }
865
866 public boolean addToCategory (String category, PublicKey key) {
867     Logger.write("VERBOSE", "DB", "addToCategory(" + category + ", ...)");
868
869     PublicKey[] members = getCategoryMembers(category);
870     if (Arrays.asList(members).contains(key)) {
871         return false;
872     }
873
874     try {
875         execute(DBStrings.addToCategory.replace("__catID__", category)
876                         .replace("__key__", Crypto.encodeKey(key)));
877     } catch (java.sql.SQLException e) {
878         Logger.write("ERROR", "DB", "SQLException: " + e);
879         return false;
880     }
881
882     return true;
883 }
884
885 public boolean removeFromCategory (Message msg) {
886     return removeFromCategory(msg.REMFROMCATgetCategory(), Crypto.decodeKey(msg.REMFROMCATgetKey()));
887 }
888
889 public boolean removeFromCategory (String category, PublicKey key) {
890     Logger.write("VERBOSE", "DB", "removeFromCategory(" + category + ", ...)");
891     try {
892         execute(DBStrings.removeFromCategory.replace("__catID__", category)
893                         .replace("__key__", Crypto.encodeKey(key)));
894     } catch (java.sql.SQLException e) {
895         Logger.write("ERROR", "DB", "SQLException: " + e);
896         return false;
897     }
898
899     return true;
900 }
901
902 public boolean like (String sig) {
903     Logger.write("VERBOSE", "DB", "like(...)");
904     try {
905         execute(DBStrings.addLike.replace("__parent__", sig)
906                         .replace("__likerKey__", Crypto.encodeKey(Crypto.getPublicKey())));
907     } catch (java.sql.SQLException e) {
908         Logger.write("ERROR", "DB", "SQLException: " + e);
909         return false;
910     }
911
912     return true;
913 }
914
915 public boolean unlike (String sig) {
916     Logger.write("VERBOSE", "DB", "like(...)");
917     try {
918         execute(DBStrings.removeLike.replace("__parent__", sig)
919                         .replace("__likerKey__", Crypto.encodeKey(Crypto.getPublicKey())));
920     } catch (java.sql.SQLException e) {
921         Logger.write("ERROR", "DB", "SQLException: " + e);
922         return false;
923     }
924
925     return true;
926 }
```

```
927 }
```

```
1 package ballmerpeak.turtlenet.server;
2
3 class DBStrings {
4     public static final String[] createDB = {
5         "CREATE TABLE tConvos (" +
6             "convoID TEXT," + //sig
7             "timeCreated TEXT," +
8             "PRIMARY KEY (convoID)" +
9         ");",
10
11        "CREATE TABLE tConvoKeys (" +
12            "pk INTEGER PRIMARY KEY AUTOINCREMENT," +
13            "convoID TEXT," +
14            "key TEXT" +
15        ");",
16
17        "CREATE TABLE tConvoMessages (" +
18            "pk INTEGER PRIMARY KEY AUTOINCREMENT," +
19            "convoID TEXT," +
20            "sendersKey TEXT," +
21            "msgText TEXT," +
22            "time TEXT" +
23        ");",
24
25        "CREATE TABLE tPost (" +
26            "sig TEXT," +
27            "msgText TEXT," +
28            "time TEXT," +
29            "recieverKey TEXT," + //person whos wall it was posted on
30            "sendersKey TEXT," +
31            "PRIMARY KEY (sig));",
32
33        "CREATE TABLE tPostVisibleTo (" +
34            "pk INTEGER PRIMARY KEY AUTOINCREMENT," +
35            "postSig TEXT," +
36            "key TEXT" +
37        ");",
38
39        "CREATE TABLE tUser (" +
40            "key TEXT," +
41            "username TEXT," +
42            "knowName INT," + //1 if we know the username for this key, 0 otherwise
43            "email TEXT," +
44            "name TEXT," +
45            "gender TEXT," +
46            "birthday TEXT," +
47            "PRIMARY KEY (key));",
48
49        "CREATE TABLE tCategory (" +
50            "catID TEXT," + //sig
51            "canSeePDATA INT," + //1 if category can see pdata, 0 otherwise
52            "PRIMARY KEY (catID));",
53
54        "CREATE TABLE tCategoryMembers (" +
55            "pk INTEGER PRIMARY KEY AUTOINCREMENT," +
56            "catID TEXT," +
57            "userKey TEXT" +
58        ");",
59
60        "CREATE TABLE tEvent (" +
61            "sig TEXT," +
62            "startTime TEXT," +
63            "endTime TEXT," +
64            "creatorKey TEXT," +
65            "accepted INT," + //1 if accepted, 0 otherwise
66            "name TEXT," +
67            "creationTime TEXT," +
68            "PRIMARY KEY (sig));",
69
70        "CREATE TABLE tClaim (" +
71            "sig TEXT," +
72            "name TEXT," +
73            "claimTime TEXT," +
74            "PRIMARY KEY (sig));",
75
76        "CREATE TABLE tLike (" +
77            "pk INTEGER PRIMARY KEY AUTOINCREMENT," +
78            "likerKey TEXT," +
79            "parent TEXT" + //sig of thing being liked
80        ");",
81
82        "CREATE TABLE tComment (" +
83            "sig TEXT," +
84            "msgText TEXT," +
85            "senderKey TEXT," +
86            "parent TEXT," + //sig of thing being commented
87            "creationTime TEXT," +
88            "PRIMARY KEY (sig));",
89
90        "CREATE TABLE tRevocations (" +
91            "key TEXT," +
92            "sig TEXT," +
93            "timeOfLeak TEXT," +
```

```

94     "creationTime TEXT,"+
95     "PRIMARY KEY (sig));",
96 };
97
98     public static final String getPPDATA
99     public static final String getWallPostSigs
100    public static final String getPost
101    sig = '_SIG_';
102    public static final String getPostSender
103    public static final String getVisibleTo
104    public static final String getConversation
105    '_SIG_';
106    public static final String getConversations
107    public static final String getKey
108    public static final String getCategories
109    public static final String getCategory
110    public static final String canSeePPDATA
111    public static final String categoriesCanSeePPDATA
112    public static final String getAllKeys
113    public static final String getMemberKeys
114    public static final String getName
115    public static final String getClaims
116    public static final String getLike
117    public static final String getComments
118    public static final String getComment
119    public static final String getRevocations
120    public static final String isRevoked
121    public static final String mostRecentWallPost
122
123    receiverKey) AS temp "+*
124
125    +
126
127    public static final String addPost
128    '_sendersKey_');";
129    public static final String addPostVisibility
130
131    public static final String addKey
132    public static final String newUsername
133    public static final String removeClaim
134    public static final String addClaim
135    '_name_','_time_');
136    public static final String addRevocation
137    ('_key_','_sig_','_time_','_creationTime_')
138    public static final String updateRevocationKey
139    public static final String addPPDATA
140    public static final String addConvo
141    '_time_');
142    public static final String addConvoParticipant
143    public static final String addMessageToConvo
144    '_time_';
145    public static final String addComment
146    '_creationTime_';
147    public static final String addLike
148    '_parent_';
149    public static final String removeLike
150    '_parent_';
151    public static final String addEvent
152    name, creationTime)+"_
153
154    //revocation stuff
155    public static final String removeMessageAccess
156    public static final String removeMessages
157    public static final String removePosts
158    public static final String removePostVisibility
159    public static final String removeUser
160    public static final String removeFromCategories
161    public static final String removeLikes
162    public static final String removeComments
163    public static final String removeEvents
164 }

= "SELECT __FIELD__ FROM tUser WHERE key = '__KEY__';";
= "SELECT sig FROM tPost WHERE recieverKey = '__KEY__';";
= "SELECT time, sig, msgText, recieverKey, sendersKey FROM tPost WHERE
= '_SIG__';";
= "SELECT sendersKey FROM tPost WHERE sig = '__SIG__';";
= "SELECT key FROM tPostVisibleTo WHERE postSig = '__SIG__';";
= "SELECT sendersKey, msgText, time FROM tConvoMessages WHERE convoID =
= '__SIG__';";
= "SELECT * FROM tConvos;";
= "SELECT key FROM tConvoKeys WHERE convoID = '__SIG__';";
= "SELECT sendersKey, time, msgText FROM tConvoMessages WHERE convoID
= '__SIG__';";
= "SELECT key FROM tUser WHERE username = '__USERNAME__';";
= "SELECT * FROM tCategory;";
= "SELECT * FROM tCategory WHERE catID = '__CATNAME__';";
= "SELECT canSeePPDATA FROM tCategory WHERE catID = '__CATID__';";
= "SELECT catID FROM tCategory WHERE canSeePPDATA = 1;";
= "SELECT key FROM tUser;";
= "SELECT userKey FROM tCategoryMembers WHERE catID = '__CATNAME__';";
= "SELECT username FROM tUser WHERE key = '__KEY__';";
= "SELECT * FROM tClaim;";
= "SELECT * FROM tLike WHERE parent = '__SIG__';";
= "SELECT * FROM tComment WHERE parent = '__PARENT__';";
= "SELECT * FROM tComment WHERE sig = '__SIG__';";
= "SELECT * FROM tRevocations;";
= "SELECT key FROM tRevocations WHERE key = '__KEY__';";
= "SELECT maxtime FROM tPost " +
"INNER JOIN " +
"(SELECT MAX(time) maxtime, recieverKey FROM tPost GROUP BY
ON tPost.recieverKey = temp.recieverKey AND tPost.time = temp.maxtime
WHERE tPost.recieverKey = '__KEY__';";

= "INSERT INTO tPost (sig, msgText, time, recieverKey, sendersKey)" +
"VALUES ('__SIG__', '__msgText__', '__time__', '__recieverKey__',
= "INSERT INTO tPostVisibleTo (postSig, key)" +
"VALUES ('__postSig__', '__key__');");
= "INSERT INTO tUser (key) VALUES ('__key__');";
= "UPDATE tUser SET username = '__name__' WHERE key = '__key__';";
= "DELETE FROM tClaim WHERE sig = '__sig__';";
= "INSERT INTO tClaim (sig, name, claimTime) VALUES ('__sig__',
= "INSERT INTO tRevocations (key, sig, timeOfLeak, creationTime) VALUES
());";
= "UPDATE tRevocations SET key = '__KEY__' WHERE sig = '__SIG__';";
= "UPDATE tUser SET __field__ = '__value__' WHERE key = '__key__';";
= "INSERT INTO tConvos (convoID, timeCreated) VALUES ('__sig__',
= "INSERT INTO tConvoKeys (convoID, key) VALUES ('__sig__', '__key__');";
= "INSERT INTO tConvoMessages (convoID, sendersKey, msgText, time)" +
"VALUES ('__convoID__', '__sendersKey__', '__msgText__',
= "INSERT INTO tComment (sig, msgText, senderKey, parent, creationTime)" +
"VALUES ('__sig__', '__msgText__', '__senderKey__', '__parent__',
= "INSERT INTO tLike (likerKey, parent) VALUES ('__likerKey__',
= "DELETE FROM tLike WHERE likerKey = '__likerKey__' AND parent =
= "INSERT INTO tEvent (sig, startTime, endTime, creatorKey, accepted,
name, creationTime)+" +
"VALUES ('__sig__', '__startTime__', '__endTime__',
= "__creatorKey__','__accepted__','__name__','__creationTime__');";
= "UPDATE tEvent SET accepted = 1 WHERE sig = '__sig__';";
= "UPDATE tEvent SET accepted = -1 WHERE sig = '__sig__';";
= "UPDATE tCategory SET canSeePPDATA = __bool__ WHERE catID =
= '__catID__';";
= "INSERT INTO tCategory (catID, canSeePPDATA) VALUES ('__catID__',
= "INSERT INTO tCategoryMembers (catID, userKey) VALUES ('__catID__',
= '__key__');";
= "DELETE FROM tCategoryMembers WHERE catId = '__catID__' AND userKey =
= '__key__';";

= "DELETE FROM tConvoKeys WHERE key = '__KEY__';";
= "DELETE FROM tConvoMessages WHERE sendersKey = '__KEY__';";
= "DELETE FROM tPost WHERE sendersKey = '__KEY__';";
= "DELETE FROM tPostVisibleTo WHERE key = '__KEY__';";
= "DELETE FROM tUser WHERE key = '__KEY__';";
= "DELETE FROM tCategoryMembers WHERE userKey = '__KEY__';";
= "DELETE FROM tLike WHERE likerKey = '__KEY__';";
= "DELETE FROM tComment WHERE senderKey = '__KEY__';";
= "DELETE FROM tEvent WHERE creatorKey = '__KEY__';";

```

```
1 //File IO
2
3 package ballmerpeak.turtlenet.server;
4 import java.io.*;
5 import ballmerpeak.turtlenet.server.Logger;
6
7 public class FIO {
8     public static byte[] readFileBytes (String filename) {
9         RandomAccessFile f = null;
10        byte[] bytes = null;
11
12        try {
13            f = new RandomAccessFile(filename, "r");
14            Long lsize = f.length();
15            int isize = (int)f.length();
16            if (lsize == isize) {
17                bytes = new byte[isize];
18                f.readFully(bytes);
19            } else {
20                Logger.write("FATAL", "FIO", filename + " is too large, could not read file.");
21            }
22            f.close();
23        } catch (IOException e) {
24            Logger.write("FATAL", "FIO", "Could not read file: " + e);
25            return bytes = null;
26        }
27
28        return bytes;
29    }
30
31    public static boolean writeFileBytes (byte[] data, String filename) {
32        FileOutputStream out;
33        try {
34            out = new FileOutputStream(new File(filename));
35            out.write(data);
36            out.close();
37            return true;
38        } catch (IOException e) {
39            Logger.write("FATAL", "FIO", "Could not write file: " + e);
40            return false;
41        }
42    }
43 }
```

```
1  /* Message Levels:
2   * UNIMPL
3   * VERBOSE - Way to much detail
4   * INFO - Normal running, useful to follow execution
5   * WARNING - Something wierd is going on, someone fucked up
6   * RED - Recoverable error (one query failing, one timeout)
7   * ERROR - Something went badly wrong
8   * FATAL - Going to crash, far more worrying if it doesn't crash
9   * CRITICAL - Fuck everything, the moon is purple
10 */
11
12 package ballmerpeak.turtlenet.server;
13
14 import java.io.*;
15 import java.util.Date;
16
17 public class Logger {
18     static boolean started = false;
19     static String path;
20     static PrintWriter log;
21
22     public static void init (String logfile) {
23         if (!started) {
24             started = true;
25             path = logfile;
26
27             try {
28                 log = new PrintWriter(new BufferedWriter(new FileWriter(path)));
29                 log.println("===== Turtlenet started at " + new Date() + "=====");
30                 log.flush();
31             } catch (Exception e) {
32                 throw new RuntimeException("ERROR: Unable to open log: " + e);
33             }
34         }
35     }
36
37     public static void close () {
38         if(started) {
39             log.println("===== Turtlenet closed at " + new Date() + "=====");
40             log.flush();
41             log.close();
42         }
43     }
44
45     public static void write (String level, String place, String s) {
46         if (started) {
47             log.println((System.currentTimeMillis()/1000L) + " " + level + getTabs(level) + place + "\t" + s);
48             log.flush(); //In case of a crash we don't want to be digging up the wrong code
49         }
50     }
51
52     private static String getTabs (String s) {
53         s = (System.currentTimeMillis()/1000L) + " " + s;
54         if (s.length() < 16) return "\t\t"; else return "\t";
55     }
56 }
```

```
1 //Can not be Message constructors because of GWT
2 //These methods can't be static like they should be because of GWT
3
4 package ballmerpeak.turtlenet.server;
5 import ballmerpeak.turtlenet.shared.Message;
6 import ballmerpeak.turtlenet.server.Crypto;
7 import java.security.*;
8
9
10 public class MessageFactory {
11     public MessageFactory(){
12     }
13
14     public Message newMessage(String cmd, String content) {
15         long timestamp = System.currentTimeMillis();
16         Message msg = new Message(cmd, content, timestamp, "");
17         msg.signature = Crypto.sign(msg);
18         return msg;
19     }
20
21     public Message newCLAIM(String username) {
22         return newMessage("CLAIM", username);
23     }
24
25     public Message newREVOKE(long time) {
26         return newMessage("REVOKE", ":"+time);
27     }
28
29     public Message newPDATA(String field, String value) {
30         return newMessage("PDATA", field + ":" + value + ";");
31     }
32
33     public Message newPDATA(String[] fields, String[] values) {
34         String content = "";
35         for (int i = 0; i < fields.length; i++) {
36             content += (values[i] + ":" + fields[i] + ";");
37         }
38         Logger.write("VERBOSE", "Msgf", "constructed pdata message: " + content);
39         return newMessage("PDATA", content);
40     }
41
42     public Message newCHAT(PublicKey[] keys) {
43         String keyString = "";
44         String delim = "";
45         for (int i = 0; i < keys.length; i++) {
46             keyString += delim + Crypto.encodeKey(keys[i]);
47             delim = ":"; /*intentional*/
48         }
49         return newMessage("CHAT", keyString);
50     }
51
52     public Message newCHAT(String[] keys) {
53         String keyString = "";
54         String delim = "";
55         for (int i = 0; i < keys.length; i++) {
56             keyString += delim + keys[i];
57             delim = ":"; /*intentional*/
58         }
59         return newMessage("CHAT", keyString);
60     }
61
62     public Message newPCHAT(String convoSig, String msg) {
63         return newMessage("PCHAT", convoSig + ":" + msg);
64     }
65
66     public Message newPOST(String msg, String wall, String[] visibleTo) {
67         String content = wall;
68         for (int i = 0; i < visibleTo.length; i++) {
69             content += (" :" + visibleTo[i]);
70         }
71         content += (" ;" + msg);
72         return newMessage("POST", content);
73     }
74
75     public Message newCMNT(String itemSig, String comment) {
76         return newMessage("CMNT", itemSig + ":" + comment);
77     }
78
79     public Message newLIKE(String itemSig) {
80         return newMessage("LIKE", itemSig);
81     }
82
83     public Message newUNLIKE(String itemSig) {
84         return newMessage("UNLIKE", itemSig);
85     }
86
87     public Message newEVNT(long start, long end, String descrip) {
88         return newMessage("EVNT", start + ":" + end + ":" + descrip);
89     }
90
91     public Message newADDCAT(String name, boolean canSeePDATA) {
92         return newMessage("ADDCAT", (canSeePDATA?"true":"false") + ":" + name);
93     }
94
95     public Message newUPDATECAT(String category, boolean value) {
```

```
94         return newMessage("UPDATECAT", (value?"true":"false") + ":" + category);
95     }
96
97     public Message newADDT0CAT(String category, String key) {
98         return newMessage("ADDTOCAT", key + ":" + category);
99     }
100
101    public Message newREMFROMCAT(String category, String key) {
102        return newMessage("REMFROMCAT", key + ":" + category);
103    }
104
105    public Message newADDKEY(String key) {
106        return newMessage("ADDKEY", key);
107    }
108 }
```

```
1 package ballmerpeak.turtlenet.server;
2
3 import ballmerpeak.turtlenet.shared.Message;
4 import java.util.Vector;
5 import java.util.Date;
6 import java.security.*;
7 import java.io.*;
8 import java.net.*;
9 import java.util.concurrent.Semaphore;
10
11 public class NetworkConnection implements Runnable {
12     public NetworkConnection (String serverurl) {
13         url      = serverurl;
14         messages = new Vector<String>();
15         lastRead = 0;
16         messageLock = new Semaphore(1);
17         connected = true;
18         tor      = true;
19
20         //parse db/lastread
21         File lastReadFile = new File("./db/lastread");
22         if (lastReadFile.exists()) {
23             try {
24                 BufferedReader reader = new BufferedReader(
25                     new FileReader(lastReadFile));
26                 lastRead = Long.parseLong(reader.readLine());
27                 Logger.write("INFO", "NetCon", "Read lastRead from file");
28             } catch (Exception e) {
29                 Logger.write("ERROR", "NetCon", "Could not read lastread from file");
30             }
31         }
32     }
33
34     public void run () {
35         Logger.write("INFO", "NetCon", "NetworkConnection started");
36         while (connected) {
37             try {
38                 Thread.sleep(1000); //update every second
39             } catch (Exception e) {
40                 Logger.write("WARNING", "NetCon", "Sleep interrupted: " + e);
41             }
42             downloadNewMessages();
43         }
44     }
45
46     public void close () {
47         Logger.write("INFO", "NetCon", "close()");
48         connected = false;
49         try {
50             File lastReadFile = new File("./db/lastread");
51
52             if (lastReadFile.exists())
53                 lastReadFile.delete();
54
55             BufferedWriter writer = new BufferedWriter(
56                 new FileWriter(lastReadFile));
57             writer.write(Long.toString(lastRead));
58             writer.close();
59             Logger.write("INFO", "NetCon", "Saved lastRead to disk");
60         } catch (Exception e) {
61             Logger.write("ERROR", "NetCon", "Unable to save lastRead: " + e);
62         }
63     }
64
65     //returns true if a message is available
66     public Boolean hasMessage () {
67         try {
68             messageLock.acquire();
69             boolean haveMessage = messages.size() >= 1;
70             messageLock.release();
71             return haveMessage;
72         } catch (Exception e) {
73             Logger.write("WARNING", "NetCon", "Acquire interrupted");
74         }
75         return false;
76     }
77
78     //get the next message in the queue, and remove it from the queue
79     public String getMessage() {
80         try {
81             messageLock.acquire();
82             String m = messages.get(0);
83             messages.removeElementAt(0);
84             messageLock.release();
85             return m;
86         } catch (Exception e) {
87             Logger.write("WARNING", "NetCon", "Acquire interrupted");
88         }
89         return new Message("NULL", "", 0, "").toString();
90     }
91
92     public long getTime () {
93         Vector<String> time = serverCmd("t");
```

```
94
95     if (time.size() == 2)
96         return Long.parseLong(time.get(0));
97     else
98         Logger.write("ERROR", "NetCon", "Couldn't retrieve time from server");
99
100    return 0;
101 }
102
103 public boolean postMessage (Message msg, PublicKey recipient) {
104     String ciphertext = Crypto.encrypt(msg, recipient, this);
105     if (!serverCmd("s " + ciphertext).get(0).equals("s"))
106         Logger.write("RED", "NetCon", "server reported failure uploading message");
107     return false;
108 } else {
109     Logger.write("INFO", "NetCon", "uploaded message: \" + msg + "\"");
110     return true;
111 }
112 }
113
114 //The only time unencrypted data is sent
115 public Boolean claimName (String name) {
116     try {
117         Message claim = new Message("CLAIM", name,
118             getTime() + Crypto.rand(0, 50), "");
119         claim.signature = Crypto.sign(claim);
120         String cmd = "c " + Crypto.Base64Encode(claim.toString().getBytes("UTF-8"));
121         if (serverCmd(cmd).get(0).equals("s")) {
122             Logger.write("INFO", "NetCon", "Claimed name: " + name);
123             Logger.write("INFO", "NetCon", "\tname: " + claim.CLAIMgetName());
124             Logger.write("INFO", "NetCon", "\ttime: " + Long.toString(claim.getTimestamp()));
125             Logger.write("INFO", "NetCon", "\tsig: " + claim.getSig());
126             return true;
127         }
128     } catch (Exception e) {
129         Logger.write("ERROR", "NetCon", "Could not register name: " + e);
130     }
131
132     Logger.write("INFO", "NetCon", "Could not register name: " + name);
133     return false;
134 }
135
136 public void downloadNewMessages () {
137     Vector<String> msgs = serverCmd("get " + lastRead);
138     lastRead = getTime();
139
140     for (int i = 0; i < msgs.size(); i++) {
141         if (!(msgs.get(i) == null) && !msgs.get(i).equals("s") && !msgs.get(i).equals("e")) {
142             try {
143                 messageLock.acquire();
144                 messages.add(msgs.get(i));
145                 messageLock.release();
146             } catch (Exception e) {
147                 Logger.write("WARNING", "NetCon", "Acquire interrupted.");
148             }
149         }
150     }
151 }
152
153 //send text to the server, receive its response
154 private Vector<String> serverCmd(String cmd) {
155     Socket s;
156     BufferedReader in;
157     PrintWriter out;
158     //if (!cmd.equals("t") && !cmd.substring(0, 4).equals("get "))
159     //    Logger.write("VERBOSE", "NetCon", "Sending command to server \" + cmd + \"");
160
161     //connect
162     try {
163         if (tor) {
164             s = new Socket(new Proxy(Proxy.Type.SOCKS,
165                 new InetSocketAddress("localhost", 9050))); //connect to Tor SOCKS proxy
166             s.connect(new InetSocketAddress(url, port)); //connect to server through Tor
167         } else {
168             s = new Socket(url, port);
169         }
170
171         in = new BufferedReader(new InputStreamReader(s.getInputStream()));
172         out = new PrintWriter(s.getOutputStream(), true);
173     } catch (Exception e) {
174         Logger.write("ERROR", "NetCon", "Could not connect to network: " + e);
175         return null;
176     }
177
178     //send command
179     out.println(cmd);
180     out.flush();
181
182     //receive output of server
183     Vector<String> output = new Vector<String>();
184     try {
185         String line = null;
186         do {
```

```
187         line = in.readLine();
188         if (line != null)
189             output.add(line);
190     } while (line != null);
191 } catch (Exception e) {
192     Logger.write("ERROR", "NetCon", "Could not read from rserver: " + e.getMessage());
193 }
194
195     //disconnect
196     try {
197         out.close();
198     } catch (Exception e) {
199         Logger.write("ERROR", "NetCon", "Could not disconnect from rserver: " + e.getMessage());
200     }
201
202     try {
203         in.close();
204     } catch (Exception e) {
205         Logger.write("ERROR", "NetCon", "Could not disconnect from rserver: " + e.getMessage());
206     }
207
208     try {
209         s.close();
210     } catch (Exception e) {
211         Logger.write("ERROR", "NetCon", "Could not close socket: " + e.getMessage());
212     }
213
214     return output;
215 }
216
217 private String url;
218 private final int port = 31415;
219 private Vector<String> messages;
220
221 private long lastRead;
222 private boolean connected;
223 private boolean tor;
224 private Semaphore messageLock;
225 }
```

```
1 //All methods ought to be static
2 //Most real parsing occurs in the Message class, this just passes commands to DB
3 package ballmerpeak.turtlenet.server;
4
5 import ballmerpeak.turtlenet.shared.Message;
6
7 public class Parser {
8     /* Useful to ID the type of message on behalf of the DB so it can use type
9      * specific get methods (e.g.: Message.PCHATgetConversationID()). Most
10     * parsing actually occurs in the Message class itself. Maybe this should
11     * be changed so parsing occurs here, e.g.: Parser.LIKEgetItemID(msg), but
12     * msg.LIKEgetItemID() is more natural.
13     */
14     public static void parse (Message msg, Database db) {
15
16         Logger.write("VERBOSE", "PARSE", "parsing message");
17
18         escape(msg);
19
20         if (db.isRevoked(db.getSignatory(msg)))
21             Logger.write("WARNING", "PARSE", "Revoked key in use, message dropped");
22         else if (msg.getCmd().equals("POST"))           //post to own wall
23             db.addPost(msg);
24         else if (msg.getCmd().equals("CLAIM"))          //claim a username
25             db.addClaim(msg);
26         else if (msg.getCmd().equals("REVOKE"))          //revoke private key
27             db.addRevocation(msg);
28         else if (msg.getCmd().equals("PDATA"))           //create or update profile data
29             db.addPDATA(msg);
30         else if (msg.getCmd().equals("CHAT"))            //establish chat
31             db.addConvo(msg);
32         else if (msg.getCmd().equals("PCHAT"))           //add message to chat
33             db.addMessageToChat(msg);
34         else if (msg.getCmd().equals("CMNT"))            //comment
35             db.addComment(msg);
36         else if (msg.getCmd().equals("LIKE"))             //like
37             db.addLike(msg);
38         else if (msg.getCmd().equals("UNLIKE"))           //like
39             db.unlike(msg.UNLIKEgetItemID());
40         else if (msg.getCmd().equals("EVNT"))             //event
41             db.addEvent(msg);
42         else if (msg.getCmd().equals("ADDCAT"))           //add category
43             db.addCategory(msg);
44         else if (msg.getCmd().equals("UPDATECAT"))        //update categories canSeePDATA
45             db.updatePDATApermission(msg);
46         else if (msg.getCmd().equals("ADDTOCAT"))         //add key to category
47             db.addToCategory(msg);
48         else if (msg.getCmd().equals("REMFROMCAT"))        //remove key from category
49             db.removeFromCategory(msg);
50         else if (msg.getCmd().equals("ADDKEY"))            //add public key
51             db.addKey(msg);
52         else if (msg.getCmd().equals("NULL"))
53             Logger.write("VERBOSE", "PARSE", "undecryptable message"); //not for us
54         else if (!msg.getCmd().equals("FPOST"))
55             Logger.write("ERROR", "PARSE", "Unknown message type: \\" + msg.getCmd() + "\\"");
56
57         if (msg.getCmd().equals("FPOST"))
58             Logger.write("WARNING", "PARSE", "FPOST is deprecated");
59     }
60
61     private static void escape (Message m) {
62         m.content = m.content.replace("\\\\", "\\");
63     }
64 }
```

```
1 package ballmerpeak.turtlenet.shared;
2
3 import ballmerpeak.turtlenet.shared.Tokenizer;
4 import java.security.*;
5 import java.io.Serializable;
6 import java.util.Arrays;
7
8 public class Message implements Serializable {
9     //You shouldn't use this, rather use MessageFactory.newMessage(command, data)
10    //GWT cannot use the factory, it shouldn't construct messages but pass their
11    // data as arguments to whatever needs it. Maybe have an async factory?
12    public Message (String cmd, String _content, long timeCreated, String RSAsig) {
13        command = cmd;
14        content = _content;
15        signature = RSAsig;
16        timestamp = timeCreated;
17    }
18
19    public Message () {
20        command = "NULL";
21        content = "";
22        signature = "";
23        timestamp = -1;
24    }
25
26    /* "POST\520adfc4>Hello, World!\123" -> new Message("POST", "Hello, World!", "520adfc4", 123) */
27    public static Message parse (String msg) {
28        String[] tokens = new String[4];
29        Tokenizer tokenizer = new Tokenizer(msg, "\\\\");
30        tokens[0] = tokenizer.nextToken(); //command
31        tokens[1] = tokenizer.nextToken(); //signature
32        tokens[2] = msg.substring(msg.indexOf("\\"), msg.indexOf("\\",0)+1)+1, msg.lastIndexOf("\\")); //message content
33        tokens[3] = msg.substring(msg.lastIndexOf("\\")+1); //timestamp
34        long ts = Long.parseLong(tokens[3]);
35
36        return new Message(tokens[0], tokens[2], ts, tokens[1]);
37    }
38
39    public String toString () {
40        return command + "\\" + signature + "\\" + content + "\\" + timestamp;
41    }
42
43    /* universal */
44    public String getCmd () {
45        return command;
46    }
47
48    public String getSig () {
49        return signature;
50    }
51
52    public String getContent () {
53        return content;
54    }
55
56    public long getTimestamp () {
57        return timestamp;
58    }
59
60    /* type specific */
61    public String POSTgetText() {
62        Tokenizer tokenizer = new Tokenizer(content, ":");
63        String[] colonPairs = new String[tokenizer.countTokens()];
64        for (int i = 0; tokenizer.hasMoreTokens(); i++)
65            colonPairs[i] = tokenizer.nextToken();
66        return colonPairs[colonPairs.length-1];
67    }
68
69    public String POSTgetWall() {
70        Tokenizer tokenizer = new Tokenizer(content, ':');
71        String[] colonPairs = new String[tokenizer.countTokens()];
72        for (int i = 0; tokenizer.hasMoreTokens(); i++)
73            colonPairs[i] = tokenizer.nextToken();
74        return colonPairs[0];
75    }
76
77    public String[] POSTgetVisibleTo() {
78        Tokenizer tokenizer = new Tokenizer(content, ':');
79        String[] colonPairs = new String[tokenizer.countTokens()];
80        for (int i = 0; tokenizer.hasMoreTokens(); i++)
81            colonPairs[i] = tokenizer.nextToken();
82        return Arrays.copyOfRange(colonPairs, 1, colonPairs.length-1);
83    }
84
85    public String CLAIMgetName() {
86        return content;
87    }
88
89    //content in form "field1:value1;field2:value2;""
90    public String[][] PDATAGetValues() {
91        //Split into colon pairs, semicolon delimiter
92        Tokenizer tokenizer = new Tokenizer(content, ';');
93        String[] colonPairs = new String[tokenizer.countTokens()];
```

```
94     for (int i = 0; tokenizer.hasMoreTokens(); i++)  
95         colonPairs[i] = tokenizer.nextToken();  
96  
97     //split into field/value pairs, colon delimiter  
98     String[][] values = new String[colonPairs.length][2];  
99     for (int i = 0; i < colonPairs.length; i++) {  
100         values[i][0] = Message.beforeColon(colonPairs[i]);  
101         values[i][1] = Message.afterColon(colonPairs[i]);  
102     }  
103  
104     return values;  
105 }  
106  
107 /* establish a chat and the people in it, without any messages */  
108 // returns an array of strings and now of keys because of GWT,  
109 // Crypto.decodeKey should be used to turn each string into a key  
110 public String[] CHATgetKeys() {  
111     Tokenizer st = new Tokenizer(content, ':');  
112     String[] keys = new String[st.countTokens()];  
113     for (int i = 0; i < keys.length; i++)  
114         keys[i] = st.nextToken();  
115     return keys;  
116 }  
117  
118 /* PCHAT adds messages to a conversation */  
119 /* returns <conversation ID, messageText> */  
120 public String PCHATgetText() {  
121     Tokenizer st = new Tokenizer(content, ':');  
122     String convoID = st.nextToken();  
123     String text = st.nextToken();  
124     return text;  
125 }  
126  
127 public String PCHATgetConversationID() {  
128     Tokenizer st = new Tokenizer(content, ':');  
129     String convoID = st.nextToken();  
130     String text = st.nextToken();  
131     return convoID;  
132 }  
133  
134 public String CMNTgetText() {  
135     Tokenizer st = new Tokenizer(content, ':');  
136     String itemID = st.nextToken();  
137     String text = st.nextToken();  
138     return text;  
139 }  
140  
141 public String CMNTgetItemID() {  
142     Tokenizer st = new Tokenizer(content, ':');  
143     String itemID = st.nextToken();  
144     String text = st.nextToken();  
145     return itemID;  
146 }  
147  
148 public String LIKEgetItemID() {  
149     return content;  
150 }  
151  
152 public String UNLIKEgetItemID() {  
153     return content;  
154 }  
155  
156 public String EVNTgetName() {  
157     Tokenizer st = new Tokenizer(content, ':');  
158     long start = Long.parseLong(st.nextToken());  
159     long end = Long.parseLong(st.nextToken());  
160     String name = st.nextToken();  
161     return name;  
162 }  
163  
164 public long EVNTgetStart() {  
165     Tokenizer st = new Tokenizer(content, ':');  
166     long start = Long.parseLong(st.nextToken());  
167     long end = Long.parseLong(st.nextToken());  
168     String name = st.nextToken();  
169     return start;  
170 }  
171  
172 public long EVNTgetEnd() {  
173     Tokenizer st = new Tokenizer(content, ':');  
174     long start = Long.parseLong(st.nextToken());  
175     long end = Long.parseLong(st.nextToken());  
176     String name = st.nextToken();  
177     return end;  
178 }  
179  
180 /* time of revocation, not timestamp of message */  
181 /* there cannot be a REVOKEgetKey due to GWT */  
182 public long REVOKEgetTime() {  
183     try {  
184         return Long.parseLong(content);  
185     } catch (Exception e) {  
186         //Invalid timestamp
```

```
187         return -1;
188     }
189 }
190
191 public String ADDCATgetName() {
192     return Message.afterColon(content);
193 }
194
195 public boolean ADDCATgetValue() {
196     return Message.beforeColon(content).equals("true");
197 }
198
199 public String UPDATECATgetName() {
200     return Message.afterColon(content);
201 }
202
203 public boolean UPDATECATgetValue() {
204     return Message.beforeColon(content).equals("true");
205 }
206
207 public String ADDTOCATgetName() {
208     return Message.afterColon(content);
209 }
210
211 public String ADDTOCATgetKey() {
212     return Message.beforeColon(content);
213 }
214
215 public String REMFROMCATgetCategory() {
216     return Message.afterColon(content);
217 }
218
219 public String REMFROMCATgetKey() {
220     return Message.beforeColon(content);
221 }
222
223 public String ADDKEYgetKey() {
224     return content;
225 }
226
227 public static String beforeColon(String s) {
228     return s.substring(0, s.indexOf(':'));
229 }
230
231 public static String afterColon(String s) {
232     return s.substring(s.indexOf(':')+1);
233 }
234
235 public String command;
236 public String content;
237 public String signature;
238 public long timestamp;
239 }
```

```
1 package ballmerpeak.turtlenet.server;
2
3 import java.security.PublicKey;
4
5 public class Friend {
6     public Friend (String _name, PublicKey _key) {
7         name = _name;
8         key = _key;
9     }
10
11    public String getName () {
12        return name;
13    }
14
15    public void setName (String nname) {
16        name = nname;
17    }
18
19    public PublicKey getKey () {
20        return key;
21    }
22
23    private String name;
24    private PublicKey key;
25 }
```

```
1 package ballmerpeak.turtlenet.server;
2
3 public class Pair<A,B> {
4     public Pair(A f, B s) {
5         first = f;
6         second = s;
7     }
8
9     public A first;
10    public B second;
11 }
```

```
1 //To reduce RPC calls
2
3 package ballmerpeak.turtlenet.shared;
4
5 import java.io.Serializable;
6
7 public class CommentDetails implements Serializable {
8     public CommentDetails () {
9         }
10
11     public CommentDetails (String _posterKey, String _posterName, String _sig, String _text, boolean _liked, Long _timestamp) {
12         posterKey = _posterKey;
13         posterName = _posterName;
14         sig = _sig;
15         text = _text;
16         liked = _liked;
17         timestamp = _timestamp;
18     }
19
20     public String posterKey;
21     public String posterName;
22     public String sig;
23     public String text;
24     public boolean liked;
25     public Long timestamp;
26 }
```

```
1 package ballmerpeak.turtlenet.shared;
2
3 import java.io.Serializable;
4
5 public class Conversation implements Serializable {
6     public Conversation () {
7         signature = "<SIGNATURE NOT KNOWN>";
8         timestamp = "0";
9         firstMessage = "<FIRST MESSAGE NOT KNOWN>";
10        users = new String[0];
11        keys = new String[0];
12    }
13
14    public Conversation (String sig, String time, String fmsg, String[] _users, String[] _keys) {
15        signature = sig;
16        timestamp = time;
17        firstMessage = fmsg;
18        users = _users;
19        keys = _keys;
20    }
21
22    public String concatNames() {
23        String names = "";
24        for (int i = 0; i < users.length; i++)
25            names += users[i] + " ";
26        return names;
27    }
28
29    public String signature;
30    public String timestamp;
31    public String firstMessage;
32    public String[] users; //usernames
33    public String[] keys; //keys[0] is the key of users[0], etc.
34 }
```

```
1 //To reduce RPC calls
2
3 package ballmerpeak.turtlenet.shared;
4
5 import java.io.Serializable;
6
7 public class PostDetails implements Serializable {
8     public PostDetails () {
9         }
10
11     public PostDetails (String _sig, boolean _liked, int _commentCount, Long _timestamp, String _posterUsername, String _text,
12     String _posterKey) {
13         sig = _sig;
14         liked = _liked;
15         commentCount = _commentCount;
16         timestamp = _timestamp;
17         posterKey = _posterKey;
18         posterUsername = _posterUsername;
19         text = _text;
20     }
21
22     public String sig;
23     public boolean liked;
24     public int commentCount;
25     public Long timestamp;
26     public String posterKey;
27     public String posterUsername;
28     public String text;
29 }
```

```
1 //Can't user java.util.StringTokenizer because of GWT
2 package ballmerpeak.turtlenet.shared;
3
4 public class Tokenizer {
5     String[] tokens;
6     int i = 0;
7
8     public Tokenizer (String s, char c) {
9         String regex = "" + c;
10        if (c == '\\')
11            regex = "\\\\";
12        tokens = s.split(regex);
13    }
14
15    public String nextToken () {
16        return tokens[i++];
17    }
18
19    public boolean hasMoreTokens () {
20        return i < tokens.length;
21    }
22
23    public int countTokens () {
24        return tokens.length;
25    }
26 }
```

```
1 package ballmerpeak.turtlenet.remoteserver;
2
3 import javax.xml.bind.DatatypeConverter;
4 import java.security.MessageDigest;
5
6 class Hasher {
7     public static String hash (String data) {
8         try {
9             MessageDigest hasher = MessageDigest.getInstance("SHA-256");
10            byte[] hash = hasher.digest(data.getBytes("UTF-8"));
11            return DatatypeConverter.printHexBinary(hash);
12        } catch (Exception e) {
13            System.out.println("SHA-256 isn't supported.");
14        }
15        return "not_a_hash";
16    }
17 }
```

Appendix H

Programmers Reference Manual

Reference Manual

Generated by Doxygen 1.8.1.2

Fri May 9 2014 08:15:29

Contents

1	Class Index	1
1.1	Class List	1
2	Class Documentation	2
2.1	ballmerpeak.turtlenet.shared.CommentDetails Class Reference	2
2.1.1	Detailed Description	2
2.2	ballmerpeak.turtlenet.shared.Conversation Class Reference	2
2.2.1	Detailed Description	3
2.3	ballmerpeak.turtlenet.server.Crypto Class Reference	3
2.3.1	Detailed Description	4
2.3.2	Member Function Documentation	4
2.3.3	Member Data Documentation	9
2.4	ballmerpeak.turtlenet.server.Database Class Reference	9
2.4.1	Detailed Description	12
2.4.2	Constructor & Destructor Documentation	12
2.4.3	Member Function Documentation	12
2.5	ballmerpeak.turtlenet.server.DBStrings Class Reference	25
2.5.1	Detailed Description	27
2.6	ballmerpeak.turtlenet.server.FIO Class Reference	27
2.6.1	Detailed Description	27
2.7	ballmerpeak.turtlenet.server.Friend Class Reference	27
2.7.1	Detailed Description	27
2.8	ballmerpeak.turtlenet.remoteserver.Hasher Class Reference	27
2.8.1	Detailed Description	28
2.9	ballmerpeak.turtlenet.server.Logger Class Reference	28
2.9.1	Detailed Description	28
2.10	ballmerpeak.turtlenet.shared.Message Class Reference	28
2.10.1	Detailed Description	29
2.11	ballmerpeak.turtlenet.server.MessageFactory Class Reference	29
2.11.1	Detailed Description	30
2.11.2	Member Function Documentation	30
2.12	ballmerpeak.turtlenet.server.NetworkConnection Class Reference	35
2.12.1	Detailed Description	36
2.12.2	Constructor & Destructor Documentation	36
2.12.3	Member Function Documentation	36
2.13	ballmerpeak.turtlenet.server.Pair< A, B > Class Reference	38
2.13.1	Detailed Description	38
2.14	ballmerpeak.turtlenet.server.Parser Class Reference	38

2.14.1 Detailed Description	39
2.15 ballmerpeak.turtlenet.shared.PostDetails Class Reference	39
2.15.1 Detailed Description	39
2.16 ballmerpeak.turtlenet.remoteserver.Server Class Reference	39
2.16.1 Detailed Description	40
2.17 ballmerpeak.turtlenet.remoteserver.Session Class Reference	40
2.17.1 Detailed Description	40
2.18 ballmerpeak.turtlenet.server.TNClient Class Reference	40
2.18.1 Detailed Description	41
2.19 ballmerpeak.turtlenet.shared.Tokenizer Class Reference	41
2.19.1 Detailed Description	41
2.20 ballmerpeak.turtlenet.server.TurtlenetImpl Class Reference	41
2.20.1 Detailed Description	43
2.20.2 Member Function Documentation	43

1 Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

ballmerpeak.turtlenet.shared.CommentDetails	2
ballmerpeak.turtlenet.shared.Conversation	2
ballmerpeak.turtlenet.server.Crypto	3
ballmerpeak.turtlenet.server.Database	9
ballmerpeak.turtlenet.server.DBStrings	25
ballmerpeak.turtlenet.server.FIO	27
ballmerpeak.turtlenet.server.Friend	27
ballmerpeak.turtlenet.remoteserver.Hasher	27
ballmerpeak.turtlenet.server.Logger	28
ballmerpeak.turtlenet.shared.Message	28
ballmerpeak.turtlenet.server.MessageFactory	29
ballmerpeak.turtlenet.server.NetworkConnection	35
ballmerpeak.turtlenet.server.Pair< A, B >	38
ballmerpeak.turtlenet.server.Parser	38
ballmerpeak.turtlenet.shared.PostDetails	39
ballmerpeak.turtlenet.remoteserver.Server	39

ballmerpeak.turtlenet.remoteserver.Session	40
ballmerpeak.turtlenet.server.TNClient	40
ballmerpeak.turtlenet.shared.Tokenizer	41
ballmerpeak.turtlenet.server.TurtlenetImpl	41

2 Class Documentation

2.1 ballmerpeak.turtlenet.shared.CommentDetails Class Reference

Inherits Serializable.

Public Member Functions

- **CommentDetails** (String _posterKey, String _posterName, String _sig, String _text, boolean _liked, Long _timestamp)

Public Attributes

- String **posterKey**
- String **posterName**
- String **sig**
- String **text**
- boolean **liked**
- Long **timestamp**

2.1.1 Detailed Description

Definition at line 7 of file CommentDetails.java.

The documentation for this class was generated from the following file:

- src/ballmerpeak/turtlenet/shared/CommentDetails.java

2.2 ballmerpeak.turtlenet.shared.Conversation Class Reference

Inherits Serializable.

Public Member Functions

- **Conversation** (String sig, String time, String fmsg, String[] _users, String[] _keys)
- String **concatNames** ()

Public Attributes

- String **signature**
- String **timestamp**
- String **firstMessage**
- String[] **users**
- String[] **keys**

2.2.1 Detailed Description

Definition at line 5 of file Conversation.java.

The documentation for this class was generated from the following file:

- src/ballmerpeak/turtlenet/shared/Conversation.java

2.3 ballmerpeak.turtlenet.server.Crypto Class Reference

Static Public Member Functions

- static Boolean **keysExist** ()
Check whether the user has a keypair.
- static void **keyGen** ()
Generate an RSA keypair.
- static boolean **encryptDB** (String password)
Encrypt local data store.
- static boolean **decryptDB** (String password)
Decrypt local data store.
- static KeyPair **getTestKey** ()
Get a keypair for testing purposes.
- static PublicKey **getPublicKey** ()
Get the current users public key.
- static PrivateKey **getPrivateKey** ()
Get the current users private key.
- static String **sign** (Message msg)
Generate the appropriate signature for a message.
- static String **sign** (Message msg, PrivateKey k)
Generate the appropriate signature for a message.
- static String **hash** (String data)
Hash a string.
- static boolean **verifySig** (Message msg, PublicKey author)
Verify a signature.
- static String **encrypt** (Message msg, PublicKey recipient, NetworkConnection connection)
Encrypt a message.
- static Message **decrypt** (String msg)
Decrypt a message.
- static String **encodeKey** (PublicKey key)
Encode a PublicKey as a String.
- static PublicKey **decodeKey** (String codedKey)
Decode a string representation of a PublicKey.
- static String **Base64Encode** (byte[] data)
Encode a byte[] as a string.
- static byte[] **Base64Decode** (String data)
Decode a string representation of a byte[] Uses base64 encoding.
- static int **rand** (int min, int max)
Get a random number.
- static byte[] **encryptBytes** (byte[] data, String key)
AES Encrypt data.
- static byte[] **decryptBytes** (byte[] data, String key)
Decrypt AES data.

Static Public Attributes

- static SecureRandom **srand**

Static Private Member Functions

- static byte[] **getAESKey** (String password)

Derives a byte[] suitable for constructing an AES key from a string.

2.3.1 Detailed Description

Definition at line 18 of file Crypto.java.

2.3.2 Member Function Documentation**2.3.2.1 static byte [] ballmerpeak.turtlenet.server.Crypto.Base64Decode (String data) [static]**

Decode a string representation of a byte[] Uses base64 encoding.

Parameters

<i>data</i>	The string to decode.
-------------	-----------------------

Returns

The byte[] data represents.

Definition at line 378 of file Crypto.java.

2.3.2.2 static String ballmerpeak.turtlenet.server.Crypto.Base64Encode (byte[] data) [static]

Encode a byte[] as a string.

Uses base64 encoding.

Parameters

<i>data</i>	The byte array to encode.
-------------	---------------------------

Returns

A string representation of data.

Definition at line 369 of file Crypto.java.

2.3.2.3 static PublicKey ballmerpeak.turtlenet.server.Crypto.decodeKey (String codedKey) [static]

Decode a string representation of a PublicKey.

Parameters

<i>codedKey</i>	The encoded public key to decode.
-----------------	-----------------------------------

Returns

The PublicKey encoded in codedKey.

Definition at line 350 of file Crypto.java.

2.3.2.4 static Message ballmerpeak.turtlenet.server.Crypto.decrypt (String msg) [static]

Decrypt a message.

Uses an RSA header with the main ciphertext being AES encrypted. There are significant speed advantages to this given most modern CPUs have AES as part of the instruction set.

Message is decrypted using the current users private key.

Parameters

<i>msg</i>	The message to decryot.
------------	-------------------------

Returns

The decrypted message.

Definition at line 294 of file Crypto.java.

2.3.2.5 static byte [] ballmerpeak.turtlenet.server.Crypto.decryptBytes (byte[] data, String key) [static]

Decrypt AES data.

Decrypts AES encrypted byte[]s using a key derived from key.

Parameters

<i>data</i>	The data to decrypt.
<i>key</i>	The string from which the key is derived.

Returns

decrypted data.

Definition at line 416 of file Crypto.java.

2.3.2.6 static boolean ballmerpeak.turtlenet.server.Crypto.decryptDB (String password) [static]

Decrypt local data store.

Decrypts all files in the db folder with AES-128. Removes the trailing .aes from the filename

Parameters

<i>password</i>	The password used to derive the AES key.
-----------------	------------------------------------------

Returns

true of success, false otherwise.

Definition at line 98 of file Crypto.java.

2.3.2.7 static String ballmerpeak.turtlenet.server.Crypto.encodeKey (PublicKey key) [static]

Encode a PublicKey as a String.

Parameters

<i>key</i>	The PublicKey to encode.
------------	--------------------------

Returns

A string representation of key.

Definition at line 337 of file Crypto.java.

2.3.2.8 static String ballmerpeak.turtlenet.server.Crypto.encrypt (Message msg, PublicKey recipient, NetworkConnection connection) [static]

Encrypt a message.

Uses an RSA header with the main ciphertext being AES encrypted. There are significant speed advantages to this given most modern CPUs have AES as part of the instruction set.

Message is encrypted using the current users private key.

Time differentials can, and have, been used to correlate otherwise anonymous messages; therefore server time is used. This is not to protect against malicious server operators, but operators ordered after the fact to provide the data they've collected.

The [NetworkConnection](#) is used to get the servers time.

Parameters

<i>msg</i>	The message to encrypt.
<i>recipient</i>	The public key that should be used to encrypt the message.
<i>connection</i>	A network connection to the server you intend to send this message to. Used to get a timestamp.

Returns

The encrypted message as a string.

Definition at line 254 of file Crypto.java.

2.3.2.9 static byte [] ballmerpeak.turtlenet.server.Crypto.encryptBytes (byte[] data, String key) [static]

AES Encrypt data.

AES encrypts a byte[] using AES and a key derived from key.

Parameters

<i>data</i>	The data to encrypt.
<i>key</i>	The string from which the key is derived.

Returns

data encrypted using AES.

Definition at line 398 of file Crypto.java.

2.3.2.10 static boolean ballmerpeak.turtlenet.server.Crypto.encryptDB (String password) [static]

Encrypt local data store.

encrypts all files in the db folder with AES-128. Renames files to <filename>.aes

Parameters

<i>password</i>	The password used to derive the AES key.
-----------------	------------------------------------------

Returns

true of success, false otherwise.

Definition at line 72 of file Crypto.java.

2.3.2.11 static byte[] ballmerpeak.turtlenet.server.Crypto.getAESKey(String password) [static], [private]

Derives a byte[] suitable for constructing an AES key from a string.

Parameters

<i>password</i>	The string from which to derive a byte[].
-----------------	-------------------------------------------

Returns

A byte[] suitable for constructing an AES key.

Definition at line 432 of file Crypto.java.

2.3.2.12 static PrivateKey ballmerpeak.turtlenet.server.Crypto.getPrivateKey() [static]

Get the current users private key.

Returns

The current users PrivateKey

Definition at line 152 of file Crypto.java.

2.3.2.13 static PublicKey ballmerpeak.turtlenet.server.Crypto.getPublicKey() [static]

Get the current users public key.

Returns

The current users PublicKey.

Definition at line 137 of file Crypto.java.

2.3.2.14 static KeyPair ballmerpeak.turtlenet.server.Crypto.getTestKey() [static]

Get a keypair for testing purposes.

Parameters

<i>bar</i>	<i>baz</i> .
------------	--------------

Returns

An RSA KeyPair

Definition at line 122 of file Crypto.java.

2.3.2.15 static String ballmerpeak.turtlenet.server.Crypto.hash(String data) [static]

Hash a string.

Uses SHA-256.

Parameters

<i>data</i>	The text to hash.
-------------	-------------------

Returns

The SHA-256 hash of the data.

Definition at line 201 of file Crypto.java.

2.3.2.16 static void ballmerpeak.turtlenet.server.Crypto.keyGen() [static]

Generate an RSA keypair.

Stores the keys in [Database.path](#) + public/private .key They are automatically encrypted by [TNClient](#).

Definition at line 37 of file Crypto.java.

2.3.2.17 static Boolean ballmerpeak.turtlenet.server.Crypto.keysExist() [static]

Check whether the user has a keypair.

Returns

true is the user has a keypair, false otherwise.

Definition at line 27 of file Crypto.java.

2.3.2.18 static int ballmerpeak.turtlenet.server.Crypto.rand(int min, int max) [static]

Get a random number.

Parameters

<i>min</i>	The minimum value to return, inclusive.
<i>max</i>	The maximum value to return, exclusive.

Returns

a random integer between min and max (inclusive.)

Definition at line 387 of file Crypto.java.

2.3.2.19 static String ballmerpeak.turtlenet.server.Crypto.sign(Message msg) [static]

Generate the appropriate signature for a message.

Returns the appropriate signature for the given message as a string. Signature is created using the current users private key.

Parameters

<i>msg</i>	The message to sign.
------------	----------------------

Returns

A signature of msg by the current users private key.

Definition at line 170 of file Crypto.java.

2.3.2.20 static String ballmerpeak.turtlenet.server.Crypto.sign(Message msg, PrivateKey k) [static]

Generate the appropriate signature for a message.

Returns the appropriate signature for the given message as a string. Signature is created using the given key.

Parameters

<i>msg</i>	The message to sign.
<i>k</i>	The key to sign with.

Returns

A signature of msg by k.

Definition at line 182 of file Crypto.java.

2.3.2.21 static boolean ballmerpeak.turtlenet.server.Crypto.verifySig (Message msg, PublicKey author) [static]

Verify a signature.

Parameters

<i>msg</i>	The message that's been signed.
<i>author</i>	The key suspected to be the author of the signature.

Returns

true is author signed msg, false otherwise.

Definition at line 216 of file Crypto.java.

2.3.3 Member Data Documentation

2.3.3.1 SecureRandom ballmerpeak.turtlenet.server.Crypto.srand [static]

Initial value:

```
new SecureRandom(
    Long.toString(
        System.currentTimeMillis())
    .getBytes())
```

Definition at line 19 of file Crypto.java.

The documentation for this class was generated from the following file:

- src/ballmerpeak/turtlenet/server/Crypto.java

2.4 ballmerpeak.turtlenet.server.Database Class Reference

Public Member Functions

- **Database** (String pw)

Construct a database object.
- void **dbCreate** ()

Creates a new database.
- boolean **dbConnect** (boolean dbexists)

Connects to an extant database.
- void **dbDisconnect** ()

Disconnects from the database.

- void `execute` (String `query`) throws java.sql.SQLException
Executes the given SQL query.
- ResultSet `query` (String `query`) throws java.sql.SQLException
Executes the given SQL query, returns the results.
- String `getPDATA` (String `field`, PublicKey `key`)
Gets the specified piece of profile data for the specified user.
- Message[] `getWallPost` (PublicKey `key`)
Get the posts on the wall of the given user.
- String `getWallPostSender` (String `sig`)
Get the key of the user who created a given post.
- Message[] `getComments` (String `sig`)
Get all comments on a given post or comment.
- Long `timeMostRecentWallPost` (PublicKey `key`)
Get the time that a given users wall was last posted on.
- boolean `isLiked` (String `sig`)
Query whether or not a given comment or post is liked.
- Conversation[] `getConversations` ()
Get all conversations you know about.
- PublicKey[] `getPeopleInConvo` (String `sig`)
Get the keys of users involved in a given conversation.
- Conversation `getConversation` (String `sig`)
Get details of a specific conversation, but not the messages therein.
- String[][] `getConversationMessages` (String `sig`)
Get all the messages from a given conversation.
- PublicKey `getKey` (String `userName`)
Retrieve the key of the specified user.
- boolean `canSeePDATA` (String `category`)
Query whether or not a given category can see your profile information.
- String[][] `getCategories` ()
Get a list of all categories.
- PublicKey[] `getCategoryMembers` (String `catID`)
Get all members of a given category.
- PublicKey[] `getVisibilityOfParent` (String `sig`)
Get all people who can see a given post or comment.
- PublicKey[] `getPostVisibleTo` (String `sig`)
Get all people who can see a given post.
- String `getName` (PublicKey `key`)
Retrieves the most recent username associated with a given key.
- PublicKey `getSignatory` (Message `m`)
Discover who sent a given message.
- boolean `addPost` (Message `post`)
Add a post.
- boolean `addKey` (Message `msg`)
Add another users public key.
- boolean `addKey` (PublicKey `k`)
Add another users public key.
- boolean `validateClaims` (PublicKey `k`)
Check username claims against a given public key.
- boolean `calcRevocationKeys` (PublicKey `k`)
Update the keys column in the key revocations table.
- boolean `addClaim` (Message `claim`)

- boolean `addRevocation` (`Message` revocation)
Add a revocation.
- boolean `isRevoked` (`PublicKey` key)
Check if a given key is revoked.
- boolean `eraseContentFrom` (`PublicKey` key)
Erase all content signed by a given key.
- boolean `addPDATA` (`Message` update)
Add new profile information.
- boolean `updatePDATA` (String field, String value, `PublicKey` k)
Update a given users profile information.
- boolean `addConvo` (`Message` convo)
Add a new conversation.
- boolean `addMessageToChat` (`Message` msg)
Add a message to an extant conversation.
- boolean `addComment` (`Message` comment)
Add a comment to a post or comment.
- boolean `addLike` (`Message` like)
Add a like.
- boolean `addEvent` (`Message` event)
Add a new event.
- boolean `acceptEvent` (String sig)
Accept an event.
- boolean `declineEvent` (String sig)
Decline an event.
- boolean `updatePDATAPermission` (`Message` msg)
Update whether or not a given category can see your profile information.
- boolean `updatePDATAPermission` (String category, boolean value)
Change whether or not a category can see your profile information.
- `PublicKey[] keysCanSeePDATA` ()
Get a list of people who can see your profile information.
- boolean `addCategory` (`Message` msg)
Add a new category.
- boolean `addCategory` (String name, boolean can_see_private_details)
Add a new category.
- boolean `addToCategory` (`Message` msg)
Add a user to a category.
- boolean `addToCategory` (String category, `PublicKey` key)
Add a user to a category.
- boolean `removeFromCategory` (`Message` msg)
Remove a user from a category.
- boolean `removeFromCategory` (String category, `PublicKey` key)
Remove a user form a category.
- boolean `like` (String sig)
Like a given post or comment.
- boolean `unlike` (String sig)
Unlike a given post or comment.

Static Public Member Functions

- static boolean **DBDirExists** ()
Test whether the directory used for storing local data exists.
- static boolean **DBExists** ()
Checks whether the database exists or not.
- static boolean **createDBDir** ()
Creates a directory to store local data.

Static Public Attributes

- static String **path** = "./db"
path to database directory

Private Attributes

- Connection **dbConnection**
- String **password** = "UNSET"
Users password, used to encrypt local data.

2.4.1 Detailed Description

Definition at line 13 of file Database.java.

2.4.2 Constructor & Destructor Documentation

2.4.2.1 ballmerpeak.turtlenet.server.Database (String *pw*)

Construct a database object.

Creates the database if it doesn't exist, if it does exit then it is decrypted. It creates a directory ([Database.path](#)) to store local data if it doesn't already exist. It then connects to the database and returns.

Parameters

<i>pw</i>	The password the database is/should be encrypted with.
-----------	--------------------------------------------------------

Definition at line 25 of file Database.java.

2.4.3 Member Function Documentation

2.4.3.1 boolean ballmerpeak.turtlenet.server.Database.acceptEvent (String *sig*)

Accept an event.

Parameters

<i>sig</i>	The signature of the event you are accepting.
------------	-----------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 960 of file Database.java.

2.4.3.2 boolean ballmerpeak.turtlenet.server.Database.addCategory (Message msg)

Add a new category.

Parameters

<i>msg</i>	The Message object representing the new category.
------------	---------------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 1043 of file Database.java.

2.4.3.3 boolean ballmerpeak.turtlenet.server.Database.addCategory (String name, boolean can_see_private_details)

Add a new category.

Parameters

<i>name</i>	The name of the new category.
<i>can_see_private_details</i>	true if the new category ought to be able to see your profile information, false otherwise.

Returns

"true" if successful, "false" otherwise.

Definition at line 1053 of file Database.java.

2.4.3.4 boolean ballmerpeak.turtlenet.server.Database.addClaim (Message claim)

Add a claim.

If a username is already claimed, forget the old name.

Parameters

<i>claim</i>	The Message object representing the claim.
--------------	--------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 726 of file Database.java.

2.4.3.5 boolean ballmerpeak.turtlenet.server.Database.addComment (Message comment)

Add a comment to a post or comment.

Parameters

<i>comment</i>	The Message object representing the comment.
----------------	----------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 898 of file Database.java.

2.4.3.6 boolean ballmerpeak.turtlenet.server.Database.addConvo (Message convo)

Add a new conversation.

Parameters

<i>convo</i>	The Message object representing the conversation.
--------------	---------------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 846 of file Database.java.

2.4.3.7 boolean ballmerpeak.turtlenet.server.Database.addEvent (Message event)

Add a new event.

Parameters

<i>event</i>	The Message object representing the event.
--------------	--------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 938 of file Database.java.

2.4.3.8 boolean ballmerpeak.turtlenet.server.Database.addKey (Message msg)

Add another users public key.

Parameters

<i>msg</i>	The Message object representing the ADDKEY message.
------------	-----------------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 624 of file Database.java.

2.4.3.9 boolean ballmerpeak.turtlenet.server.Database.addKey (PublicKey k)

Add another users public key.

Parameters

<i>k</i>	The PublicKey to add.
----------	-----------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 632 of file Database.java.

2.4.3.10 boolean ballmerpeak.turtlenet.server.Database.addLike (Message like)

Add a like.

Parameters

<i>like</i>	The Message object representing the like.
-------------	-------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 920 of file Database.java.

2.4.3.11 boolean ballmerpeak.turtlenet.server.Database.addMessageToChat (Message *msg*)

Add a message to an extant conversation.

Parameters

<i>msg</i>	The Message object representing the new message to be added.
------------	--------------------------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 869 of file Database.java.

2.4.3.12 boolean ballmerpeak.turtlenet.server.Database.addPDATA (Message *update*)

Add new profile information.

Parameters

<i>update</i>	The Message object representing the update.
---------------	---------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 808 of file Database.java.

2.4.3.13 boolean ballmerpeak.turtlenet.server.Database.addPost (Message *post*)

Add a post.

Parameters

<i>post</i>	The Message object representing the post.
-------------	-------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 601 of file Database.java.

2.4.3.14 boolean ballmerpeak.turtlenet.server.Database.addRevocation (Message *revocation*)

Add a revocation.

Parameters

<i>revocation</i>	The Message object representing the revocation.
-------------------	-------------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 748 of file Database.java.

2.4.3.15 boolean ballmerpeak.turtlenet.server.Database.addToCategory (Message msg)

Add a user to a category.

Parameters

<i>msg</i>	The Message object representing the addition of a user to a category.
------------	-----------------------------------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 1070 of file Database.java.

2.4.3.16 boolean ballmerpeak.turtlenet.server.Database.addToCategory (String category, PublicKey key)

Add a user to a category.

Parameters

<i>category</i>	The category to add a user to.
<i>key</i>	The key of the user who is being added to the specified category.

Returns

"true" if successful, "false" otherwise.

Definition at line 1079 of file Database.java.

2.4.3.17 boolean ballmerpeak.turtlenet.server.Database.calcRevocationKeys (PublicKey k)

Update the keys column in the key revocations table.

Called automatically after a new key is added to ensure it isn't revoked.

Parameters

<i>k</i>	The new key.
----------	--------------

Returns

"true" if successful, "false" otherwise.

Definition at line 693 of file Database.java.

2.4.3.18 boolean ballmerpeak.turtlenet.server.Database.canSeePDATA (String category)

Query whether or not a given category can see your profile information.

Parameters

<i>category</i>	The name of the category in question.
-----------------	---------------------------------------

Returns

true is the people in the category can see your profile information, false otherwise.

Definition at line 432 of file Database.java.

2.4.3.19 static boolean ballmerpeak.turtlenet.server.Database.createDBDir() [static]

Creates a directory to store local data.

The directory is that specified in [Database.path](#).

Definition at line 52 of file Database.java.

2.4.3.20 boolean ballmerpeak.turtlenet.server.Database.dbConnect (boolean dbexists)

Connects to an extant database.

Called automatically by the constructor.

Parameters

<code>dbexists</code>	true if a database already exists, false otherwise.
-----------------------	-----------------------------------------------------

Returns

returns true if successful at connecting, false otherwise.

Definition at line 77 of file Database.java.

2.4.3.21 void ballmerpeak.turtlenet.server.Database.dbCreate()

Creates a new database.

Called automatically by the constructor.

Definition at line 59 of file Database.java.

2.4.3.22 static boolean ballmerpeak.turtlenet.server.Database.DBDirExists() [static]

Test whether the directory used for storing local data exists.

The directory is specified by [Database.path](#).

Returns

true is the directory exists, false otherwise.

Definition at line 35 of file Database.java.

2.4.3.23 static boolean ballmerpeak.turtlenet.server.Database.DBExists() [static]

Checks whether the database exists or not.

Returns

true if the database exists, false otherwise.

Definition at line 43 of file Database.java.

2.4.3.24 boolean ballmerpeak.turtlenet.server.Database.declineEvent (String sig)

Decline an event.

Parameters

<i>sig</i>	The signature of the event you are declining.
------------	-----------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 976 of file Database.java.

2.4.3.25 boolean ballmerpeak.turtlenet.server.Database.eraseContentFrom (PublicKey key)

Erase all content signed by a given key.

Parameters

<i>key</i>	The key whose data is being expurgated.
------------	-----------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 782 of file Database.java.

2.4.3.26 void ballmerpeak.turtlenet.server.Database.execute (String query) throws java.sql.SQLException

Executes the given SQL query.

Warning

Throws a java.sql.SQLException on failure.

Parameters

<i>query</i>	The query to execute.
--------------	-----------------------

Definition at line 111 of file Database.java.

2.4.3.27 String [][] ballmerpeak.turtlenet.server.Database.getCategories ()

Get a list of all categories.

Returns

The names of each category and if it can see your profile info. Data is in this format: `{"friends", "false"}, {"family", "true"}, etc.`

Definition at line 451 of file Database.java.

2.4.3.28 PublicKey [] ballmerpeak.turtlenet.server.Database.getCategoryMembers (String catID)

Get all members of a given category.

If the category given is "all" then all known people are returned.

Parameters

<i>catID</i>	The name of the category of which you want to know the members.
--------------	-----------------------------------------------------------------

Returns

An array of PublicKeys containing the key of every user in the specified category.

Definition at line 479 of file Database.java.

2.4.3.29 Message [] ballmerpeak.turtlenet.server.Database.getComments (String *sig*)

Get all comments on a given post or comment.

Parameters

<i>sig</i>	The signature of the item whose comments one desires.
------------	-------------------------------------------------------

Returns

An array of Messages, ordered chronologically so that element 0 is the oldest comment on the item.

Definition at line 247 of file Database.java.

2.4.3.30 Conversation ballmerpeak.turtlenet.server.Database.getConversation (String *sig*)

Get details of a specific conversation, but not the messages therein.

Parameters

<i>sig</i>	The signature of the conversation you want details about.
------------	-----------------------------------------------------------

Returns

A conversation object with the details of the specified conversation.

Definition at line 343 of file Database.java.

2.4.3.31 String [][] ballmerpeak.turtlenet.server.Database.getConversationMessages (String *sig*)

Get all the messages from a given conversation.

Parameters

<i>sig</i>	The signature of the conversation you want the messages of.
------------	-------------------------------------------------------------

Returns

The messages in the specified conversation, in chronological order (i.e.: Element 0 is the oldest message in the conversation). Data is in this format: {{username, time, msg}, {username, time, msg}, etc.}

Definition at line 378 of file Database.java.

2.4.3.32 Conversation [] ballmerpeak.turtlenet.server.Database.getConversations ()

Get all conversations you know about.

Returns

An array of all conversations you know about in no particular order.

Definition at line 304 of file Database.java.

2.4.3.33 PublicKey ballmerpeak.turtlenet.server.Database.getKey (String *userName*)

Retrieve the key of the specified user.

Parameters

<i>userName</i>	The username of the user which you wish to know the key of.
-----------------	-------------------------------------------------------------

Returns

The key of the specified user. Returns "–INVALID KEYSTRING–" if no key is known.

Definition at line 404 of file Database.java.

2.4.3.34 String ballmerpeak.turtlenet.server.Database.getName (PublicKey *key*)

Retrieves the most recent username associated with a given key.

Parameters

<i>key</i>	The public key of the user whose username you wish to know.
------------	-------------------------------------------------------------

Returns

The most recent username of the user with the given public key. "<no username>" is returned if no username is known.

Definition at line 559 of file Database.java.

2.4.3.35 String ballmerpeak.turtlenet.server.Database.getPDATA (String *field*, PublicKey *key*)

Gets the specified piece of profile data for the specified user.

Parameters

<i>field</i>	The name of the field the value of which you wish to retrieve. valid options are: email, name, gender, and birthday.
<i>key</i>	The key of the user which you wish to retrieve data about.

Returns

The value of the specified field for the specified user. Returns "<no value>" if no value is known.

Definition at line 165 of file Database.java.

2.4.3.36 PublicKey [] ballmerpeak.turtlenet.server.Database.getPeopleInConvo (String *sig*)

Get the keys of users involved in a given conversation.

Parameters

<i>sig</i>	The signature of the conversation being examined.
------------	---------------------------------------------------

Returns

An array of PublicKeys containing the keys of every user who is in the specified conversation.

Definition at line 324 of file Database.java.

2.4.3.37 PublicKey [] ballmerpeak.turtlenet.server.Database.getPostVisibleTo (String *sig*)

Get all people who can see a given post.

Parameters

<i>sig</i>	The signature of the post being considered.
------------	---------------------------------------------

Returns

An array of PublicKeys which contains the key of every user which is able to decrypt and view the given post.

Definition at line 539 of file Database.java.

2.4.3.38 PublicKey ballmerpeak.turtlenet.server.Database.getSignatory (Message *m*)

Discover who sent a given message.

Derived from the signautre on the message and known public keys.

Parameters

<i>m</i>	The message whose author you wish to determine.
----------	-------------------------------------------------

Returns

The public key of the user who wrote *m*. null is returned if no signatory can be found.

Definition at line 583 of file Database.java.

2.4.3.39 PublicKey [] ballmerpeak.turtlenet.server.Database.getVisibilityOfParent (String *sig*)

Get all people who can see a given post or comment.

The visibility of comments is the same as that of their parents.

Parameters

<i>sig</i>	The signature of the post or comment being considered.
------------	--------------------------------------------------------

Returns

An array of PublicKeys which contains the key of every user which is able to decrypt and view the given comment or post.

Definition at line 512 of file Database.java.

2.4.3.40 Message [] ballmerpeak.turtlenet.server.Database.getWallPost (PublicKey *key*)

Get the posts on the wall of the given user.

Parameters

<i>key</i>	The public key of the user the wall of whom you are interested in.
------------	--------------------------------------------------------------------

Returns

An array of Messages, ordered chronologically so that element 0 is the oldest post on their wall.

Definition at line 193 of file Database.java.

2.4.3.41 String ballmerpeak.turtlenet.server.Database.getWallPostSender (String *sig*)

Get the key of the user who created a given post.

Parameters

<i>sig</i>	The signature of the post.
------------	----------------------------

Returns

The key of the user who created the post. Returns "<POST DOESN'T EXIST>" if the post doesn't exist, may also return "ERROR" if there is an SQL error.

Definition at line 228 of file Database.java.

2.4.3.42 boolean ballmerpeak.turtlenet.server.Database.isLiked (String *sig*)

Query whether or not a given comment or post is liked.

Parameters

<i>sig</i>	The signature of the comment or post being examined.
------------	------------------------------------------------------

Returns

true if the item is liked, false otherwise.

Definition at line 287 of file Database.java.

2.4.3.43 boolean ballmerpeak.turtlenet.server.Database.isRevoked (PublicKey *key*)

Check if a given key is revoked.

Parameters

<i>key</i>	The key being checked.
------------	------------------------

Returns

"true" if revoked, "false" otherwise.

Definition at line 767 of file Database.java.

2.4.3.44 PublicKey [] ballmerpeak.turtlenet.server.Database.keysCanSeePDATA ()

Get a list of people who can see your profile information.

Returns

An array of PublicKeys containing every key which is able to see your profile information.

Definition at line 1019 of file Database.java.

2.4.3.45 boolean ballmerpeak.turtlenet.server.Database.like (String *sig*)

Like a given post or comment.

Parameters

<i>sig</i>	The signature of the comment or post to like.
------------	-----------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 1128 of file Database.java.

2.4.3.46 ResultSet ballmerpeak.turtlenet.server.Database.query (String query) throws java.sql.SQLException

Executes the given SQL query, returns the results.

Warning

Throws a java.sql.SQLException on failure.

Parameters

<i>query</i>	The query to execute.
--------------	-----------------------

Returns

A ResultSet, the rows returned by the query.

Definition at line 138 of file Database.java.

2.4.3.47 boolean ballmerpeak.turtlenet.server.Database.removeFromCategory (Message msg)

Remove a user from a category.

Parameters

<i>msg</i>	The Message object representing the removal of a user from a category.
------------	------------------------------------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 1102 of file Database.java.

2.4.3.48 boolean ballmerpeak.turtlenet.server.Database.removeFromCategory (String category, PublicKey key)

Remove a user form a category.

Parameters

<i>category</i>	The name of the category the user is being removed from.
<i>key</i>	The key of the user to be removed from the category.

Returns

"true" if successful, "false" otherwise.

Definition at line 1111 of file Database.java.

2.4.3.49 Long ballmerpeak.turtlenet.server.Database.timeMostRecentWallPost (PublicKey key)

Get the time that a given users wall was last posted on.

Parameters

<i>key</i>	The public key of the user the wall of whom you are interested in.
------------	--------------------------------------------------------------------

Returns

The number of milliseconds from midnight january first 1970 to the time of the most recent post placed on the specified users wall.

Definition at line 271 of file Database.java.

2.4.3.50 boolean ballmerpeak.turtlenet.server.Database.unlike (String *sig*)

Unlike a given post or comment.

Parameters

<i>sig</i>	The signature of the comment or post to like.
------------	-----------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 1145 of file Database.java.

2.4.3.51 boolean ballmerpeak.turtlenet.server.Database.updatePDATA (String *field*, String *value*, PublicKey *k*)

Update a given users profile information.

Parameters

<i>field</i>	The name of the field the value of which you wish to update. valid options are: email, name, gender, and birthday.
<i>value</i>	The desired value.
<i>k</i>	The key of the user whos profile information is being updated.

Returns

"true" if successful, "false" otherwise.

Definition at line 827 of file Database.java.

2.4.3.52 boolean ballmerpeak.turtlenet.server.Database.updatePDATAPermission (Message *msg*)

Update whether or not a given category can see your profile information.

Parameters

<i>msg</i>	The Message object representing the update.
------------	---------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 992 of file Database.java.

2.4.3.53 boolean ballmerpeak.turtlenet.server.Database.updatePDATAPermission (String *category*, boolean *value*)

Change whether or not a category can see your profile information.

Parameters

<i>category</i>	The name of the category you wish to update.
<i>value</i>	The desired value, true to allow the category to see your profile data, false to forbid it.

Returns

"true" if successful, "false" otherwise.

Definition at line 1002 of file Database.java.

2.4.3.54 boolean ballmerpeak.turtlenet.server.Database.validateClaims (PublicKey *k*)

Check username claims against a given public key.

Called automatically after a new key is added to ensure that old username claims are recognised.

Parameters

<i>k</i>	The key to be considered.
----------	---------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 654 of file Database.java.

The documentation for this class was generated from the following file:

- src/ballmerpeak/turtlenet/server/Database.java

2.5 ballmerpeak.turtlenet.server.DBStrings Class Reference**Static Public Attributes**

- static final String[] **createDB**
- static final String **getPDATA** = "SELECT __FIELD__ FROM tUser WHERE key = '__KEY__';"
- static final String **getWallPostSigs** = "SELECT sig FROM tPost WHERE recieverKey = '__KEY__';"
- static final String **getPost** = "SELECT time, sig, msgText, recieverKey, sendersKey FROM tPost WHERE sig = '__SIG__';"
- static final String **getPostSender** = "SELECT sendersKey FROM tPost WHERE sig = '__SIG__';"
- static final String **getVisibleTo** = "SELECT key FROM tPostVisibleTo WHERE postSig = '__SIG__';"
- static final String **getConversation** = "SELECT sendersKey, msgText, time FROM tConvoMessages WHERE convoid = '__SIG__';"
- static final String **getConversations** = "SELECT * FROM tConvos;"
- static final String **getConversationMembers** = "SELECT key FROM tConvoKeys WHERE convoid = '__SIG__';"
- static final String **getConversationMessages** = "SELECT sendersKey, time, msgText FROM tConvoMessages WHERE convoid = '__SIG__';"
- static final String **getKey** = "SELECT key FROM tUser WHERE username = '__USERNAME__';"
- static final String **getCategories** = "SELECT * FROM tCategory;"
- static final String **getCategory** = "SELECT * FROM tCategory WHERE catID = '__CATNAME__';"
- static final String **canSeePDATA** = "SELECT canSeePDATA FROM tCategory WHERE catID = '__CATID__';"
- static final String **categoriesCanSeePDATA** = "SELECT catID FROM tCategory WHERE canSeePDATA = 1;"
- static final String **getAllKeys** = "SELECT key FROM tUser;"

- static final String **getMemberKeys** = "SELECT userKey FROM tCategoryMembers WHERE catID = '__CATID__';"
- static final String **getName** = "SELECT username FROM tUser WHERE key = '__KEY__';"
- static final String **getClaims** = "SELECT * FROM tClaim;"
- static final String **getLike** = "SELECT * FROM tLike WHERE parent = '__SIG__';"
- static final String **getComments** = "SELECT * FROM tComment WHERE parent = '__PARENT__';"
- static final String **getComment** = "SELECT * FROM tComment WHERE sig = '__SIG__';"
- static final String **getRevocations** = "SELECT * FROM tRevocations;"
- static final String **isRevoked** = "SELECT key FROM tRevocations WHERE key = '__KEY__';"
- static final String **mostRecentWallPost** = "WHERE tPost.receiverKey = '__KEY__';"
- static final String **addPost** = "VALUES ('__SIG__', '__msgText__', '__time__', '__receiverKey__', '__sendersKey__');"
- static final String **addPostVisibility** = "VALUES ('__postSig__', '__key__');"
- static final String **addKey** = "INSERT INTO tUser (key) VALUES ('__key__');"
- static final String **newUsername** = "UPDATE tUser SET username = '__name__' WHERE key = '__key__';"
- static final String **removeClaim** = "DELETE FROM tClaim WHERE sig = '__sig__';"
- static final String **addClaim** = "INSERT INTO tClaim (sig, name, claimTime) VALUES ('__sig__', '__name__', '__time__');"
- static final String **addRevocation** = "INSERT INTO tRevocations (key, sig, timeOfLeak, creationTime) VALUES ('__key__', '__sig__', '__time__', '__creationTime__');"
- static final String **updateRevocationKey** = "UPDATE tRevocations SET key = '__KEY__' WHERE sig = '__SIG__';"
- static final String **addPDATA** = "UPDATE tUser SET __field__ = '__value__' WHERE key = '__key__';"
- static final String **addConvo** = "INSERT INTO tConvos (convоВID, timeCreated) VALUES ('__sig__', '__time__');"
- static final String **addConvoParticipant** = "INSERT INTO tConvoKeys (convоВID, key) VALUES ('__sig__', '__key__');"
- static final String **addMessageToConvo** = "VALUES ('__convоВID__', '__sendersKey__', '__msgText__', '__time__');"
- static final String **addComment** = "VALUES ('__sig__', '__msgText__', '__senderKey__', '__parent__', '__creationTime__');"
- static final String **addLike** = "INSERT INTO tLike (likerKey, parent) VALUES ('__likerKey__', '__parent__');"
- static final String **removeLike** = "DELETE FROM tLike WHERE likerKey = '__likerKey__' AND parent = '__parent__';"
- static final String **addEvent** = "VALUES ('__sig__', '__startTime__', '__endTime__', '__creatorKey__', '__accepted__', '__name__', '__creationTime__');"
- static final String **acceptEvent** = "UPDATE tEvent SET accepted = 1 WHERE sig = '__sig__';"
- static final String **declineEvent** = "UPDATE tEvent SET accepted = -1 WHERE sig = '__sig__';"
- static final String **updatePDATAPermission** = "UPDATE tCategory SET canSeePDATA = __bool__ WHERE catID = '__catID__'"
- static final String **addCategory** = "INSERT INTO tCategory (catID, canSeePDATA) VALUES ('__catID__', '__canSeePDATA__');"
- static final String **addToCategory** = "INSERT INTO tCategoryMembers (catID, userKey) VALUES ('__catID__', '__key__');"
- static final String **removeFromCategory** = "DELETE FROM tCategoryMembers WHERE catID = '__catID__' AND userKey = '__key__';"
- static final String **removeMessageAccess** = "DELETE FROM tConvoKeys WHERE key = '__KEY__';"
- static final String **removeMessages** = "DELETE FROM tConvoMessages WHERE sendersKey = '__KEY__';"
- static final String **removePosts** = "DELETE FROM tPost WHERE sendersKey = '__KEY__';"
- static final String **removePostVisibility** = "DELETE FROM tPostVisibleTo WHERE key = '__KEY__';"
- static final String **removeUser** = "DELETE FROM tUser WHERE key = '__KEY__';"
- static final String **removeFromCategories** = "DELETE FROM tCategoryMembers WHERE userKey = '__KEY__';"
- static final String **removeLikes** = "DELETE FROM tLike WHERE likerKey = '__KEY__';"
- static final String **removeComments** = "DELETE FROM tComment WHERE senderKey = '__KEY__';"
- static final String **removeEvents** = "DELETE FROM tEvent WHERE creatorKey = '__KEY__';"

2.5.1 Detailed Description

Definition at line 3 of file DBStrings.java.

The documentation for this class was generated from the following file:

- src/ballmerpeak/turtlenet/server/DBStrings.java

2.6 ballmerpeak.turtlenet.server.FIO Class Reference

Static Public Member Functions

- static byte[] **readFileBytes** (String filename)
- static boolean **writeFileBytes** (byte[] data, String filename)

2.6.1 Detailed Description

Definition at line 7 of file FIO.java.

The documentation for this class was generated from the following file:

- src/ballmerpeak/turtlenet/server/FIO.java

2.7 ballmerpeak.turtlenet.server.Friend Class Reference

Public Member Functions

- **Friend** (String _name, PublicKey _key)
- String **getName** ()
- void **setName** (String name)
- PublicKey **getKey** ()

Private Attributes

- String **name**
- PublicKey **key**

2.7.1 Detailed Description

Definition at line 5 of file Friend.java.

The documentation for this class was generated from the following file:

- src/ballmerpeak/turtlenet/server/Friend.java

2.8 ballmerpeak.turtlenet.remoteserver.Hasher Class Reference

Static Public Member Functions

- static String **hash** (String data)

2.8.1 Detailed Description

Definition at line 6 of file Hasher.java.

The documentation for this class was generated from the following file:

- src/ballmerpeak/turtlenet/remoteserver/Hasher.java

2.9 ballmerpeak.turtlenet.server.Logger Class Reference

Static Public Member Functions

- static void **init** (String logfile)
- static void **close** ()
- static void **write** (String level, String place, String s)

Static Package Attributes

- static boolean **started** = false
- static String **path**
- static PrintWriter **log**

Static Private Member Functions

- static String **getTabs** (String s)

2.9.1 Detailed Description

Definition at line 17 of file Logger.java.

The documentation for this class was generated from the following file:

- src/ballmerpeak/turtlenet/server/Logger.java

2.10 ballmerpeak.turtlenet.shared.Message Class Reference

Inherits Serializable.

Public Member Functions

- **Message** (String cmd, String _content, long timeCreated, String RSAsig)
- String **toString** ()
- String **getCmd** ()
- String **getSig** ()
- String **getContent** ()
- long **getTimestamp** ()
- String **POSTgetText** ()
- String **POSTgetWall** ()
- String[] **POSTgetVisibleTo** ()
- String **CLAIMgetName** ()
- String[][] **PDATAgetValues** ()
- String[] **CHATgetKeys** ()
- String **PCHATgetText** ()

- String **PCHATgetConversationID** ()
- String **CMNTgetText** ()
- String **CMNTgetItemID** ()
- String **LIKEgetitemID** ()
- String **UNLIKEgetitemID** ()
- String **EVNTgetName** ()
- long **EVNTgetStart** ()
- long **EVNTgetEnd** ()
- long **REVOKEgetTime** ()
- String **ADDCATgetName** ()
- boolean **ADDCATgetValue** ()
- String **UPDATECATgetName** ()
- boolean **UPDATECATgetValue** ()
- String **ADDTOCATgetName** ()
- String **ADDTOCATgetKey** ()
- String **REMFROMCATgetCategory** ()
- String **REMFROMCATgetKey** ()
- String **ADDKEYgetKey** ()

Static Public Member Functions

- static [Message parse](#) (String msg)
- static String **beforeColon** (String s)
- static String **afterColon** (String s)

Public Attributes

- String **command**
- String **content**
- String **signature**
- long **timestamp**

2.10.1 Detailed Description

Definition at line 8 of file Message.java.

The documentation for this class was generated from the following file:

- src/ballmerpeak/turtlenet/shared/Message.java

2.11 ballmerpeak.turtlenet.server.MessageFactory Class Reference

Public Member Functions

- [Message newMessage](#) (String cmd, String content)
Create a new message.
- [Message newCLAIM](#) (String username)
Create a new claim message.
- [Message newREVOKE](#) (long time)
Create a new revoke message.
- [Message newPDATA](#) (String field, String value)
Create a new pdata message.
- [Message newPDATA](#) (String[] fields, String[] values)

- Create a new pdata message.*
- [Message newCHAT](#) (PublicKey[] keys)

Create a message that creates a new conversation.
 - [Message newCHAT](#) (String[] keys)

Create a message that creates a new conversation.
 - [Message newPCHAT](#) (String convoSig, String msg)

Create a message that adds a post to an extant conversation.
 - [Message newPOST](#) (String msg, String wall, String[] visibleTo)

Create a message that adds a post to an wall.
 - [Message newCMNT](#) (String itemSig, String comment)

Create a message that adds a comment to a comment or post.
 - [Message newLIKE](#) (String itemSig)

Create a message that adds a like to a comment or post.
 - [Message newUNLIKE](#) (String itemSig)

Create a message that removes a like from a comment or post.
 - [Message newEVNT](#) (long start, long end, String descrip)

Create a message that adds a new event.
 - [Message newADDCAT](#) (String name, boolean canSeePDATA)

Create a message that adds a new category.
 - [Message newUPDATECAT](#) (String category, boolean value)

Create a message that updates where a category can see your profile information.
 - [Message newADDTOTCAT](#) (String category, String key)

Create a message that adds a user to an extant category.
 - [Message newREMFROMCAT](#) (String category, String key)

Create a message that removes a user from an extant category.
 - [Message newADDKEY](#) (String key)

Create a message that adds a key to the database.

2.11.1 Detailed Description

Definition at line 10 of file MessageFactory.java.

2.11.2 Member Function Documentation

2.11.2.1 Message ballmerpeak.turtlenet.server.MessageFactory.newADDCAT (String name, boolean canSeePDATA)

Create a message that adds a new category.

Calculates timestamp and signautre for you.

Parameters

<i>name</i>	The name of the new category.
<i>canSeePDATA</i>	true is the new categroy should be able to see your profile information, false otherwise.

Returns

A message that adds a new category as specified with correct timestamp and signature.

Definition at line 169 of file MessageFactory.java.

2.11.2.2 Message ballmerpeak.turtlenet.server.MessageFactory.newADDKEY (String key)

Create a message that adds a key to the database.

Calculates timestamp and signautre for you.

Parameters

<code>key</code>	The key of the user to be added to the database.
------------------	--------------------------------------------------

Returns

A message that adds the specified user to the database with correct timestamp and signature.

Definition at line 208 of file MessageFactory.java.

2.11.2.3 Message ballmerpeak.turtlenet.server.MessageFactory.newADDTOCAT (String *category*, String *key*)

Create a message that adds a user to an extant category.

Calculates timestamp and signautre for you.

Parameters

<code>category</code>	The name of the category.
<code>key</code>	The key of the user to be added to the category.

Returns

A message that adds the specified user to the specified category with correct timestamp and signature.

Definition at line 189 of file MessageFactory.java.

2.11.2.4 Message ballmerpeak.turtlenet.server.MessageFactory.newCHAT (PublicKey[] *keys*)

Create a message that creates a new conversation.

Calculates timestamp and signautre for you.

Parameters

<code>keys</code>	The people in the conversation.
-------------------	---------------------------------

Returns

A message that creates the specified conversation with correct timestamp and signature.

Definition at line 74 of file MessageFactory.java.

2.11.2.5 Message ballmerpeak.turtlenet.server.MessageFactory.newCHAT (String[] *keys*)

Create a message that creates a new conversation.

Calculates timestamp and signautre for you.

Parameters

<code>keys</code>	The people in the conversation, encoded as strings.
-------------------	-----------------------------------------------------

Returns

A message that creates the specified conversation with correct timestamp and signature.

Definition at line 89 of file MessageFactory.java.

2.11.2.6 Message ballmerpeak.turtlenet.server.MessageFactory.newCLAIM (String *username*)

Create a new claim message.

Calculates timestamp and signature for you.

Parameters

<i>username</i>	The name to claim.
-----------------	--------------------

Returns

A message claiming the given username with correct timestamp and signature.

Definition at line 32 of file MessageFactory.java.

2.11.2.7 Message ballmerpeak.turtlenet.server.MessageFactory.newCMNT (String *itemSig*, String *comment*)

Create a message that adds a comment to a comment or post.

Calculates timestamp and signature for you.

Parameters

<i>comment</i>	The text of the comment.
<i>itemSig</i>	The signature of the post or comment you want to add a comment to.

Returns

A message that adds the specified comment to the specified post or comment with correct timestamp and signature.

Definition at line 130 of file MessageFactory.java.

2.11.2.8 Message ballmerpeak.turtlenet.server.MessageFactory.newEVNT (long *start*, long *end*, String *descrip*)

Create a message that adds a new event.

Calculates timestamp and signature for you.

Parameters

<i>start</i>	The timestamp (in ms since midnight jan 1st 1970) of the start of the event.
<i>end</i>	The timestamp (in ms since midnight jan 1st 1970) of the end of the event.
<i>descrip</i>	A description of the event.

Returns

A message that adds a new event with correct timestamp and signature.

Definition at line 159 of file MessageFactory.java.

2.11.2.9 Message ballmerpeak.turtlenet.server.MessageFactory.newLIKE (String *itemSig*)

Create a message that adds a like to a comment or post.

Calculates timestamp and signature for you.

Parameters

<i>itemSig</i>	The signature of the post or comment you want to add a like to.
----------------	-----------------------------------------------------------------

Returns

A message that adds a like to the specified post or comment with correct timestamp and signature.

Definition at line 139 of file MessageFactory.java.

2.11.2.10 Message ballmerpeak.turtlenet.server.MessageFactory.newMessage (String cmd, String content)

Create a new message.

Calculates timestamp and signautre for you.

Parameters

<i>cmd</i>	An appropriate value for the cmd field, e.g.: POST, ADDKEY.
<i>content</i>	The main body of the message.

Returns

A message with correct timestamp and signature.

Definition at line 20 of file MessageFactory.java.

2.11.2.11 Message ballmerpeak.turtlenet.server.MessageFactory.newPCHAT (String convoSig, String msg)

Create a message that adds a post to an extant conversation.

Calculates timestamp and signautre for you.

Parameters

<i>convoSig</i>	The conversation to add a message to.
<i>msg</i>	The text of the message to add.

Returns

A message that adds the specified message to the specified conversation with correct timestamp and signature.

Definition at line 105 of file MessageFactory.java.

2.11.2.12 Message ballmerpeak.turtlenet.server.MessageFactory.newPDATA (String field, String value)

Create a new pdata message.

Calculates timestamp and signautre for you.

Parameters

<i>field</i>	The field you wish to set.
<i>value</i>	The value you wish to give that field.

Returns

A message updating your profile information with correct timestamp and signature.

Definition at line 51 of file MessageFactory.java.

2.11.2.13 Message ballmerpeak.turtlenet.server.MessageFactory.newPDATA (String[] fields, String[] values)

Create a new pdata message.

Calculates timestamp and signautre for you.

Parameters

<i>fields</i>	The fields you wish to set.
<i>values</i>	The values you wish to give the respective field.

Returns

A message updating your profile information with correct timestamp and signature.

Definition at line 61 of file MessageFactory.java.

2.11.2.14 Message ballmerpeak.turtlenet.server.MessageFactory.newPOST (String *msg*, String *wall*, String[] *visibleTo*)

Create a message that adds a post to an wall.

Calculates timestamp and signautre for you.

Parameters

<i>msg</i>	The text of the post.
<i>wall</i>	The key of the wall to add a post to.
<i>visibleTo</i>	The keys of everyone who is able to see the post.

Returns

A message that adds the specified post to the specified wall with correct timestamp and signature.

Definition at line 116 of file MessageFactory.java.

2.11.2.15 Message ballmerpeak.turtlenet.server.MessageFactory.newREMFROMCAT (String *category*, String *key*)

Create a message that removes a user from an extant category.

Calculates timestamp and signautre for you.

Parameters

<i>category</i>	The name of the category.
<i>key</i>	The key of the user to be removed from the category.

Returns

A message that removes the specified user from the specified category with correct timestamp and signature.

Definition at line 199 of file MessageFactory.java.

2.11.2.16 Message ballmerpeak.turtlenet.server.MessageFactory.newREVOKE (long *time*)

Create a new revoke message.

Calculates timestamp and signautre for you.

Parameters

<i>time</i>	The time from which you key ought not be trusted.
-------------	---------------------------------------------------

Returns

A message revoking your key with correct timestamp and signature.

Definition at line 41 of file MessageFactory.java.

2.11.2.17 Message ballmerpeak.turtlenet.server.MessageFactory.newUNLIKE (String itemSig)

Create a message that removes a like from a comment or post.

Calculates timestamp and signautre for you.

Parameters

<i>itemSig</i>	The signature of the post or comment you want to remove a like from.
----------------	----------------------------------------------------------------------

Returns

A message that removes a like from the specified post or comment with correct timestamp and signature.

Definition at line 148 of file MessageFactory.java.

2.11.2.18 Message ballmerpeak.turtlenet.server.MessageFactory.newUPDATECAT (String category, boolean value)

Create a message that updates where a category can see your profile information.

Calculates timestamp and signautre for you.

Parameters

<i>name</i>	The name of the category.
<i>canSeePDATA</i>	true is the new categoroy should be able to see your profile information, false otherwise.

Returns

A message that updates the specified category with correct timestamp and signature.

Definition at line 179 of file MessageFactory.java.

The documentation for this class was generated from the following file:

- src/ballmerpeak/turtlenet/server/MessageFactory.java

2.12 ballmerpeak.turtlenet.server.NetworkConnection Class Reference

Public Member Functions

- **NetworkConnection** (String serverurl)
Construct a network connection that connects to the given URL.
- void **run** ()
Used for constructing a network thread.
- void **close** ()
Shutdown the network connection.
- Boolean **hasMessage** ()
Checks for new messages.
- String **getMessage** ()
Get a message.
- long **getTime** ()
Get the server time.
- boolean **postMessage** (Message msg, PublicKey recipient)
Post a Message object over the network.
- Boolean **claimName** (String name)
Claims a username on the network.
- void **downloadNewMessages** ()
Download new messages.

Private Member Functions

- Vector< String > **serverCmd** (String cmd)
Send a command to the server.

Private Attributes

- String **url**
- final int **port** = 31415
Pi is awesome.
- Vector< String > **messages**
- long **lastRead**
- boolean **connected**
- boolean **tor**
- Semaphore **messageLock**

2.12.1 Detailed Description

Definition at line 11 of file NetworkConnection.java.

2.12.2 Constructor & Destructor Documentation

2.12.2.1 ballmerpeak.turtlenet.server.NetworkConnection.NetworkConnection (String *serverurl*)

Construct a network connection that connects to the given URL.

Parameters

<i>serverurl</i>	The URI to connect to.
------------------	------------------------

Definition at line 15 of file NetworkConnection.java.

2.12.3 Member Function Documentation

2.12.3.1 Boolean ballmerpeak.turtlenet.server.NetworkConnection.claimName (String *name*)

Claims a username on the network.

Usernames must be unique, this is enforced by the server. Thread safe.

Warning

Extant usernames are considered public information. This is the only plaintext sent in the system.

Parameters

<i>name</i>	The name to claim.
-------------	--------------------

Returns

true if succes, false otherwie.

Definition at line 153 of file NetworkConnection.java.

2.12.3.2 void ballmerpeak.turtlenet.server.NetworkConnection.close()

Shutdown the network connection.

Saves the last time at which new messages were fetched to disk. Thread safe.

Definition at line 56 of file NetworkConnection.java.

2.12.3.3 void ballmerpeak.turtlenet.server.NetworkConnection.downloadNewMessages()

Download new messages.

Automatically called every second by [run\(\)](#). Thread safe.

Definition at line 178 of file NetworkConnection.java.

2.12.3.4 String ballmerpeak.turtlenet.server.NetworkConnection.getMessage()

Get a message.

Get the next message in the queue, and remove it from the queue. Thread safe.

Returns

The oldest message in the stack of messages received.

Definition at line 96 of file NetworkConnection.java.

2.12.3.5 long ballmerpeak.turtlenet.server.NetworkConnection.getTime()

Get the server time.

Millisecond timestamps can, and have, been used to identify users. We therefore recommend always using server time. (Obviously other methods are in place to hide true network latency.) Thread safe.

Returns

The number of milliseconds since midnight january first 1970, according to the server.

Definition at line 117 of file NetworkConnection.java.

2.12.3.6 Boolean ballmerpeak.turtlenet.server.NetworkConnection.hasMessage()

Checks for new messages.

Thread safe.

Returns

returns true if a message is available, false otherwise

Definition at line 79 of file NetworkConnection.java.

2.12.3.7 boolean ballmerpeak.turtlenet.server.NetworkConnection.postMessage(Message msg, PublicKey recipient)

Post a Message object over the network.

Only viewable by recipient. Thread safe.

Parameters

<i>recipient</i>	The person you are sending the message to.
------------------	--------------------------------------------

Returns

returns true on success, false otherwise.

Definition at line 134 of file NetworkConnection.java.

2.12.3.8 void ballmerpeak.turtlenet.server.NetworkConnection.run()

Used for constructing a network thread.

Fetches new messages every second. Thread safe.

Definition at line 40 of file NetworkConnection.java.

2.12.3.9 Vector<String> ballmerpeak.turtlenet.server.NetworkConnection.serverCmd(String cmd) [private]

Send a command to the server.

Parameters

<i>cmd</i>	The text to send to the server.
------------	---------------------------------

Returns

The servers response, one string per line, element 0 is the topmost (first) line sent by the server.

Definition at line 200 of file NetworkConnection.java.

The documentation for this class was generated from the following file:

- src/ballmerpeak/turtlenet/server/NetworkConnection.java

2.13 ballmerpeak.turtlenet.server.Pair< A, B > Class Reference**Public Member Functions**

- **Pair** (A f, B s)

Public Attributes

- A **first**
- B **second**

2.13.1 Detailed Description

Definition at line 3 of file Pair.java.

The documentation for this class was generated from the following file:

- src/ballmerpeak/turtlenet/server/Pair.java

2.14 ballmerpeak.turtlenet.server.Parser Class Reference**Static Public Member Functions**

- static void **parse** ([Message](#) msg, [Database](#) db)

Static Private Member Functions

- static void **escape** ([Message m](#))

2.14.1 Detailed Description

Definition at line 7 of file Parser.java.

The documentation for this class was generated from the following file:

- src/ballmerpeak/turtlenet/server/Parser.java

2.15 ballmerpeak.turtlenet.shared.PostDetails Class Reference

Inherits Serializable.

Public Member Functions

- **PostDetails** (String _sig, boolean _liked, int _commentCount, Long _timestamp, String _posterUsername, String _text, String _posterKey)

Public Attributes

- String **sig**
- boolean **liked**
- int **commentCount**
- Long **timestamp**
- String **posterKey**
- String **posterUsername**
- String **text**

2.15.1 Detailed Description

Definition at line 7 of file PostDetails.java.

The documentation for this class was generated from the following file:

- src/ballmerpeak/turtlenet/shared/PostDetails.java

2.16 ballmerpeak.turtlenet.remoteserver.Server Class Reference

Static Public Member Functions

- static void **start** (int port)
- static void **shutdown** ()
- static void **main** (String[] argv)

Static Public Attributes

- static String **shutdownPassword** = "SHUTDOWN 83eea84d472df09f5e64468996fdff0e"

Static Private Attributes

- static ServerSocket **socket**
- static boolean **running** = true

2.16.1 Detailed Description

Definition at line 10 of file Server.java.

The documentation for this class was generated from the following file:

- src/ballmerpeak/turtlenet/remoteserver/Server.java

2.17 ballmerpeak.turtlenet.remoteserver.Session Class Reference

Public Member Functions

- void **run** ()
- void **execute** (String cmd, BufferedReader in, PrintWriter out)

Package Functions

- **Session** (Socket s)

Private Member Functions

- long **getTimestamp** (File f)
- Boolean **userExists** (String name)

Private Attributes

- Socket **client**

2.17.1 Detailed Description

Definition at line 53 of file Server.java.

The documentation for this class was generated from the following file:

- src/ballmerpeak/turtlenet/remoteserver/Server.java

2.18 ballmerpeak.turtlenet.server.TNClient Class Reference

Public Member Functions

- **TNClient** (String pw)
- void **run** ()

Public Attributes

- `NetworkConnection connection`
- Thread `networkThread`
- `Database db = null`
- String `password = "NOT SET"`
- boolean `running = true`
- boolean `dbReady = false`

2.18.1 Detailed Description

Definition at line 6 of file TNClient.java.

The documentation for this class was generated from the following file:

- src/ballmerpeak/turtlenet/server/TNClient.java

2.19 ballmerpeak.turtlenet.shared.Tokenizer Class Reference

Public Member Functions

- `Tokenizer (String s, char c)`
- String `nextToken ()`
- boolean `hasMoreTokens ()`
- int `countTokens ()`

Package Attributes

- String[] `tokens`
- int `i = 0`

2.19.1 Detailed Description

Definition at line 4 of file Tokenizer.java.

The documentation for this class was generated from the following file:

- src/ballmerpeak/turtlenet/shared/Tokenizer.java

2.20 ballmerpeak.turtlenet.server.TurtlenetImpl Class Reference

Inherits RemoteServiceServlet, and Turtlenet.

Public Member Functions

- String `startTN (String password)`
Starts Turtlenet client on the backend.
- String `stopTN ()`
Stops Turtlenet client on the backend.
- String `isFirstTime ()`
Check if this is the first time Turtlenet has been run.
- String `register (String username, String password)`
Register on the network.

- String [getMyUsername \(\)](#)
Retrieves the users username from the database.
- String [getUsername \(String key\)](#)
Retrieves the most recent username associated with a given key.
- String [getMyPDATA \(String field\)](#)
Gets the specified piece of profile data for the current user.
- String [getPDATA \(String field, String key\)](#)
Gets the specified piece of profile data for the specified user.
- String [getMyKey \(\)](#)
Retrieve the key of the current user.
- String [getKey \(String username\)](#)
Retrieve the key of the specified user.
- String[][] [getCategories \(\)](#)
Get a list of all categories.
- String[][] [getPeople \(\)](#)
Get all people you know about.
- Conversation[] [getConversations \(\)](#)
Get all conversations you know about.
- Conversation [getConversation \(String sig\)](#)
Get details of a specific conversation, but not the messages therein.
- String[][] [getConversationMessages \(String sig\)](#)
Get all the messages from a given conversation.
- String[][] [getCategoryMembers \(String category\)](#)
Get all members of a given category.
- PostDetails[] [getWallPosts \(String key\)](#)
Get the posts on the wall of the given user.
- CommentDetails[] [getComments \(String parent\)](#)
Get the details of all comments on a given post or comment.
- Long [timeMostRecentWallPost \(String key\)](#)
Get the time that a given users wall was last posted on.
- Long [getConvoLastUpdated \(String sig\)](#)
Get the time that a given conversation was last posted in.
- Long [getPostLastCommented \(String sig\)](#)
Get the time that a given comment or post was last commented on.
- String [claimUsername \(String uname\)](#)
Register a new username on the network.
- String [updatePDATA \(String field, String value\)](#)
Change your profile information.
- String [updatePDATAPermission \(String category, boolean value\)](#)
Change whether or not a category can see your profile information.
- String[] [createCHAT \(String\[\] keys\)](#)
Create a new conversation.
- String [addMessageToCHAT \(String text, String sig\)](#)
Add a message to a conversation.
- String [like \(String sig\)](#)
Add a like to a post or comment.
- String [unlike \(String sig\)](#)
Remove a like from a post or comment.
- String [addCategory \(String name\)](#)
Add a category.
- String [addToCategory \(String group, String key\)](#)

- Add a user to a category.
String [sendPDATA](#) (String key)
Send profile information to the specified key.
- String [removeFromCategory](#) (String group, String key)
Remove a user from a category.
- String [addKey](#) (String key)
Add a public key.
- String [addPost](#) (String wallKey, String categoryVisibleTo, String msg)
Add a post to a specified wall.
- String [addComment](#) (String parent, String text)
Add a comment to a specified comment or post.
- String [revokeMyKey](#) ()
Revoke the current users key.

Package Attributes

- **TNCClient c = null**
Turtlenet client that runs in the background on the backend.

2.20.1 Detailed Description

Definition at line 15 of file TurtlenetImpl.java.

2.20.2 Member Function Documentation

2.20.2.1 String ballmerpeak.turtlenet.server.TurtlenetImpl.addCategory (String name)

Add a category.

By default the category cannot see your profile information.

Parameters

<i>name</i>	The name of the category you wish to create.
-------------	----------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 480 of file TurtlenetImpl.java.

2.20.2.2 String ballmerpeak.turtlenet.server.TurtlenetImpl.addComment (String parent, String text)

Add a comment to a specified comment or post.

Commenting does not require the permission of the person who posted the item you are commenting.

Parameters

<i>parent</i>	The signature of the post or comment that you wish to comment on.
<i>text</i>	The text of the comment you wish to make.

Returns

"true" if successful, "false" otherwise.

Definition at line 598 of file TurtlenetImpl.java.

2.20.2.3 String ballmerpeak.turtlenet.server.TurtlenetImpl.addKey (String key)

Add a public key.

Parameters

<i>key</i>	The key you wish to add.
------------	--------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 555 of file TurtlenetImpl.java.

2.20.2.4 String ballmerpeak.turtlenet.server.TurtlenetImpl.addMessageToCHAT (String text, String sig)

Add a message to a conversation.

Parameters

<i>text</i>	The text of your message.
<i>sig</i>	The signature of the conversation you wish to add a message to.

Returns

"true" if successful, "false" otherwise.

Definition at line 416 of file TurtlenetImpl.java.

2.20.2.5 String ballmerpeak.turtlenet.server.TurtlenetImpl.addPost (String wallKey, String categoryVisibleTo, String msg)

Add a post to a specified wall.

Posting on another users wall does not require their permission.

Parameters

<i>wallKey</i>	The key of the user whos wall you want to post on.
<i>categoryVisible-To</i>	The name of the category of people who may see the post.
<i>msg</i>	The text of the post you wish to make.

Returns

"true" if successful, "false" otherwise.

Definition at line 570 of file TurtlenetImpl.java.

2.20.2.6 String ballmerpeak.turtlenet.server.TurtlenetImpl.addToCategory (String group, String key)

Add a user to a category.

Parameters

<i>group</i>	The name of the category you wish to add a user to.
<i>key</i>	The key you wish to add to the category.

Returns

"true" if successful, "false" otherwise.

Definition at line 494 of file TurtlenetImpl.java.

2.20.2.7 String ballmerpeak.turtlenet.server.TurtlenetImpl.claimUsername (String *uname*)

Register a new username on the network.

Parameters

<i>uname</i>	The desired username.
--------------	-----------------------

Returns

"true" if successful, "false" otherwise. Usernames must be unique and the server enforces this.

Definition at line 309 of file TurtlenetImpl.java.

2.20.2.8 String [] ballmerpeak.turtlenet.server.TurtlenetImpl.createCHAT (String[] *keys*)

Create a new conversation.

Parameters

<i>keys</i>	The keys of each person you wish to include in the conversation.
-------------	------------------------------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 369 of file TurtlenetImpl.java.

2.20.2.9 String [][] ballmerpeak.turtlenet.server.TurtlenetImpl.getCategories ()

Get a list of all categories.

Returns

The names of each category and if it can see your profile info. Data is in this format: `[{"friends", "false"}, {"family", "true"}, etc.]`

Definition at line 149 of file TurtlenetImpl.java.

2.20.2.10 String [][] ballmerpeak.turtlenet.server.TurtlenetImpl.getCategoryMembers (String *category*)

Get all members of a given category.

If the category given is "all" then all known people are returned.

Parameters

<i>category</i>	The name of the category of which you want to know the members.
-----------------	-----------------------------------------------------------------

Returns

The username and key of each member of the specified category. Data is in this format: `[{"bob", "bobs_key"}, {"john", "johns_key"}, etc.]`

Definition at line 206 of file TurtlenetImpl.java.

2.20.2.11 CommentDetails [] ballmerpeak.turtlenet.server.TurtlenetImpl.getComments (String parent)

Get the details of all comments on a given post or comment.

Parameters

<code>parent</code>	The signature of the item whose comments one desires.
---------------------	-------------------------------------------------------

Returns

An array of CommentDetails, ordered chronologically so that element 0 is the oldest comment on the item.

Definition at line 247 of file TurtlenetImpl.java.

2.20.2.12 Conversation ballmerpeak.turtlenet.server.TurtlenetImpl.getConversation (String sig)

Get details of a specific conversation, but not the messages therein.

Parameters

<code>sig</code>	The signature of the conversation you want details about.
------------------	-----------------------------------------------------------

Returns

A conversation object with the details of the specified conversation.

Definition at line 184 of file TurtlenetImpl.java.

2.20.2.13 String [][] ballmerpeak.turtlenet.server.TurtlenetImpl.getConversationMessages (String sig)

Get all the messages from a given conversation.

Parameters

<code>sig</code>	The signature of the conversation you want the messages of.
------------------	-------------------------------------------------------------

Returns

The messages in the specified conversation, in chronological order (i.e.: Element 0 is the oldest message in the conversation). Data is in this format: {{username, time, msg}, {username, time, msg}, etc.}

Definition at line 195 of file TurtlenetImpl.java.

2.20.2.14 Conversation [] ballmerpeak.turtlenet.server.TurtlenetImpl.getConversations ()

Get all conversations you know about.

Returns

An array of all conversations you know about in no particular order.

Definition at line 166 of file TurtlenetImpl.java.

2.20.2.15 Long ballmerpeak.turtlenet.server.TurtlenetImpl.getConvoLastUpdated (String sig)

Get the time that a given conversation was last posted in.

Parameters

<code>sig</code>	The signature of the conversation being examined.
------------------	---------------------------------------------------

Returns

The number of milliseconds from midnight january first 1970 to the time of the most recent message in the specified conversation.

Definition at line 285 of file TurtlenetImpl.java.

2.20.2.16 String ballmerpeak.turtlenet.server.TurtlenetImpl.getKey (String *username*)

Retrieve the key of the specified user.

Parameters

<i>username</i>	The username of the user which you wish to know the key of.
-----------------	-------------------------------------------------------------

Returns

The key of the specified user. Returns "–INVALID KEYSTRING–" if no key is known.

Definition at line 140 of file TurtlenetImpl.java.

2.20.2.17 String ballmerpeak.turtlenet.server.TurtlenetImpl.getMyKey ()

Retrieve the key of the current user.

Returns

The key of the current user.

Definition at line 130 of file TurtlenetImpl.java.

2.20.2.18 String ballmerpeak.turtlenet.server.TurtlenetImpl.getMyPDATA (String *field*)

Gets the specified piece of profile data for the current user.

Parameters

<i>field</i>	The name of the field the value of which you wish to retrieve. valid options are: email, name, gender, and birthday.
--------------	----------------------------------------------------------------------------------------------------------------------

Returns

The value of the specified field for the current user. Returns "<no value>" if no value is known.

Definition at line 110 of file TurtlenetImpl.java.

2.20.2.19 String ballmerpeak.turtlenet.server.TurtlenetImpl.getMyUsername ()

Retrieves the users username from the database.

Returns

the users username.

Definition at line 87 of file TurtlenetImpl.java.

2.20.2.20 String ballmerpeak.turtlenet.server.TurtlenetImpl.getPDATA (String *field*, String *key*)

Gets the specified piece of profile data for the specified user.

Parameters

<i>field</i>	The name of the field the value of which you wish to retrieve. valid options are: email, name, gender, and birthday.
<i>key</i>	The key of the user which you wish to retrieve data about.

Returns

The value of the specified field for the specified user. Returns "<no value>" if no value is known.

Definition at line 122 of file TurtlenetImpl.java.

2.20.2.21 String [][] ballmerpeak.turtlenet.server.TurtlenetImpl.getPeople ()

Get all people you know about.

Returns

The usernames and public keys of everyone you know about. Data is in this format: {"bob", "bobs_key"}, {"john", "johns_key"}, etc.}

Definition at line 158 of file TurtlenetImpl.java.

2.20.2.22 Long ballmerpeak.turtlenet.server.TurtlenetImpl.getPostLastCommented (String *sig*)

Get the time that a given comment or post was last commented on.

Parameters

<i>sig</i>	The signature of the comment or post being examined.
------------	------------------------------------------------------

Returns

The number of milliseconds from midnight january first 1970 to the time of the most recent comment was posted on the given comment or post.

Definition at line 299 of file TurtlenetImpl.java.

2.20.2.23 String ballmerpeak.turtlenet.server.TurtlenetImpl.getUsername (String *key*)

Retrieves the most recent username associated with a given key.

Parameters

<i>key</i>	The public key of the user whose username you wish to know.
------------	-------------------------------------------------------------

Returns

The most recent username of the user with the given public key. "<no username>" is returned if no username is known.

Definition at line 97 of file TurtlenetImpl.java.

2.20.2.24 PostDetails [] ballmerpeak.turtlenet.server.TurtlenetImpl.getWallPosts (String *key*)

Get the posts on the wall of the given user.

Parameters

<i>key</i>	The public key of the user the wall of whom you are interested in.
------------	--------------------------------------------------------------------

Returns

An array of PostDetails, ordered chronologically so that element 0 is the oldest post on their wall.

Definition at line 224 of file TurtlenetImpl.java.

2.20.2.25 String ballmerpeak.turtlenet.server.TurtlenetImpl.isFirstTime()

Check if this is the first time Turtlenet has been run.

Returns

"true" if this is the first time turtlenet has been run, "false" otherwise.

Definition at line 46 of file TurtlenetImpl.java.

2.20.2.26 String ballmerpeak.turtlenet.server.TurtlenetImpl.like(String sig)

Add a like to a post or comment.

Parameters

<i>sig</i>	The signature of the post or comment you wish to add a like to.
------------	-----------------------------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 439 of file TurtlenetImpl.java.

2.20.2.27 String ballmerpeak.turtlenet.server.TurtlenetImpl.register(String username, String password)

Register on the network.

Parameters

<i>username</i>	The desired username.
<i>password</i>	The desired password (used for local encryption, not remote).

Returns

"true" if successful, "false" otherwise. Usernames must be unique and the server enforces this.

Definition at line 56 of file TurtlenetImpl.java.

2.20.2.28 String ballmerpeak.turtlenet.server.TurtlenetImpl.removeFromCategory(String group, String key)

Remove a user from a category.

Parameters

<i>group</i>	The name of the category you wish to remove a person from.
<i>key</i>	The public key of the person you wish to be removed.

Returns

"true" if successful, "false" otherwise.

Definition at line 544 of file TurtlenetImpl.java.

2.20.2.29 String ballmerpeak.turtlenet.server.TurtlenetImpl.revokeMyKey ()

Revoke the current users key.

This marks the current users key as untrusted. All people whose key they have will be informed. This cannot be publically broadcast to all users because despite the nature of misdirection employed it would be fairly easy for the server operators to identify and suppress the revocation message.

Returns

"true" if successful, "false" otherwise.
Only people whose keys the user has added will be informed of the revocation.
This erases the users account and local database.

Returns

"true" if successful, "false" otherwise.

Definition at line 627 of file TurtlenetImpl.java.

2.20.2.30 String ballmerpeak.turtlenet.server.TurtlenetImpl.sendPDATA (String key)

Send profile information to the specified key.

This is a one off thing, the user will not be automatically kept abreast of new profile information.

Parameters

<i>key</i>	The public key of the user you wish to send profile information to.
------------	---------------------------------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 531 of file TurtlenetImpl.java.

2.20.2.31 String ballmerpeak.turtlenet.server.TurtlenetImpl.startTN (String password)

Starts Turtlenet client on the backend.

Returns

"true" if successful, "false" otherwise.

Definition at line 21 of file TurtlenetImpl.java.

2.20.2.32 String ballmerpeak.turtlenet.server.TurtlenetImpl.stopTN ()

Stops Turtlenet client on the backend.

Returns

"true" if successful, "false" otherwise.

Definition at line 37 of file TurtlenetImpl.java.

2.20.2.33 Long ballmerpeak.turtlenet.server.TurtlenetImpl.timeMostRecentWallPost (String key)

Get the time that a given users wall was last posted on.

Parameters

<i>key</i>	The public key of the user the wall of whom you are interested in.
------------	--------------------------------------------------------------------

Returns

The number of milliseconds from midnight january first 1970 to the time of the most recent post placed on the specified users wall.

Definition at line 276 of file TurtlenetImpl.java.

2.20.2.34 String ballmerpeak.turtlenet.server.TurtlenetImpl.unlike (String sig)

Remove a like from a post or comment.

Parameters

<i>sig</i>	The signature of the post or comment you wish to remove a like from.
------------	----------------------------------------------------------------------

Returns

"true" if successful, "false" otherwise.

Definition at line 459 of file TurtlenetImpl.java.

2.20.2.35 String ballmerpeak.turtlenet.server.TurtlenetImpl.updatePDATA (String field, String value)

Change your profile information.

Parameters

<i>field</i>	The name of the field the value of which you wish to update. valid options are: email, name, gender, and birthday.
<i>value</i>	The desired value.

Returns

"true" if successful, "false" otherwise.

Definition at line 324 of file TurtlenetImpl.java.

2.20.2.36 String ballmerpeak.turtlenet.server.TurtlenetImpl.updatePDATApermission (String category, boolean value)

Change whether or not a category can see your profile information.

Parameters

<i>category</i>	The name of the category you wish to update.
<i>value</i>	The desired value, true to allow the category to see your profile data, false to forbid it.

Returns

"true" if successful, "false" otherwise.

Definition at line 344 of file TurtlenetImpl.java.

The documentation for this class was generated from the following file:

- src/ballmerpeak/turtlenet/server/TurtlenetImpl.java

Index

acceptEvent
 ballmerpeak::turtlenet::server::Database, 12
addCategory
 ballmerpeak::turtlenet::server::Database, 12, 13
 ballmerpeak::turtlenet::server::TurtlenetImpl, 43
addClaim
 ballmerpeak::turtlenet::server::Database, 13
addComment
 ballmerpeak::turtlenet::server::Database, 13
 ballmerpeak::turtlenet::server::TurtlenetImpl, 43
addConvo
 ballmerpeak::turtlenet::server::Database, 13
addEvent
 ballmerpeak::turtlenet::server::Database, 14
addKey
 ballmerpeak::turtlenet::server::Database, 14
 ballmerpeak::turtlenet::server::TurtlenetImpl, 44
addLike
 ballmerpeak::turtlenet::server::Database, 14
addMessageToCHAT
 ballmerpeak::turtlenet::server::TurtlenetImpl, 44
addMessageToChat
 ballmerpeak::turtlenet::server::Database, 14
addPDATA
 ballmerpeak::turtlenet::server::Database, 15
addPost
 ballmerpeak::turtlenet::server::Database, 15
 ballmerpeak::turtlenet::server::TurtlenetImpl, 44
addRevocation
 ballmerpeak::turtlenet::server::Database, 15
addToCategory
 ballmerpeak::turtlenet::server::Database, 15, 16
 ballmerpeak::turtlenet::server::TurtlenetImpl, 44

ballmerpeak.turtlenet.remoteserver.Hasher, 27
ballmerpeak.turtlenet.remoteserver.Server, 39
ballmerpeak.turtlenet.remoteserver.Session, 40
ballmerpeak.turtlenet.server.Crypto, 2
ballmerpeak.turtlenet.server.DBStrings, 25
ballmerpeak.turtlenet.server.Database, 9
ballmerpeak.turtlenet.server.FIO, 27
ballmerpeak.turtlenet.server.Friend, 27
ballmerpeak.turtlenet.server.Logger, 28
ballmerpeak.turtlenet.server.MessageFactory, 29
ballmerpeak.turtlenet.server.NetworkConnection, 35
ballmerpeak.turtlenet.server.Pair< A, B >, 38
ballmerpeak.turtlenet.server.Parser, 38
ballmerpeak.turtlenet.server.TNClient, 40
ballmerpeak.turtlenet.server.TurtlenetImpl, 41
ballmerpeak.turtlenet.shared.CommentDetails, 1
ballmerpeak.turtlenet.shared.Conversation, 2
ballmerpeak.turtlenet.shared.Message, 28
ballmerpeak.turtlenet.shared.PostDetails, 39
ballmerpeak.turtlenet.shared.Tokenizer, 41
ballmerpeak::turtlenet::server::Crypto
 Base64Decode, 4
 Base64Encode, 4
 decodeKey, 4
 decrypt, 4
 decryptBytes, 5
 decryptDB, 5
 encodeKey, 5
 encrypt, 5
 encryptBytes, 6
 encryptDB, 6
 getAESKey, 6
 getPrivateKey, 7
 getPublicKey, 7
 getTestKey, 7
 hash, 7
 keyGen, 7
 keysExist, 7
 rand, 8
 sign, 8
 srand, 9
 verifySig, 9

ballmerpeak::turtlenet::server::Database
 acceptEvent, 12
 addCategory, 12, 13
 addClaim, 13
 addComment, 13
 addConvo, 13
 addEvent, 14
 addKey, 14
 addLike, 14
 addMessageToChat, 14
 addPDATA, 15
 addPost, 15
 addRevocation, 15
 addToCategory, 15, 16
 calcRevocationKeys, 16
 canSeePDATA, 16
 createDBDir, 16
 DBDirExists, 17
 DBExists, 17
 Database, 12
 dbConnect, 16
 dbCreate, 17
 declineEvent, 17
 eraseContentFrom, 17
 execute, 18
 getCategories, 18
 getCategoryMembers, 18
 getComments, 18
 getConversation, 19
 getConversationMessages, 19
 getConversations, 19
 getKey, 19
 getName, 20
 getPDATA, 20

getPeopleInConvo, 20
getPostVisibleTo, 20
getSignatory, 21
getVisibilityOfParent, 21
getWallPost, 21
getWallPostSender, 21
isLiked, 22
isRevoked, 22
keysCanSeePDATA, 22
like, 22
query, 23
removeFromCategory, 23
timeMostRecentWallPost, 23
unlike, 24
updatePDATA, 24
updatePDATAPermission, 24
validateClaims, 25
ballmerpeak::turtlenet::server::MessageFactory
 newADDCAT, 30
 newADDKEY, 30
 newADDTOCAT, 31
 newCHAT, 31
 newCLAIM, 31
 newCMNT, 32
 newEVNT, 32
 newLIKE, 32
 newMessage, 33
 newPCHAT, 33
 newPDATA, 33
 newPOST, 34
 newREMFROMCAT, 34
 newREVOKE, 34
 newUNLIKE, 34
 newUPDATECAT, 35
ballmerpeak::turtlenet::server::NetworkConnection
 claimName, 36
 close, 37
 downloadNewMessages, 37
 getMessage, 37
 getTime, 37
 hasMessage, 37
 NetworkConnection, 36
 postMessage, 37
 run, 38
 serverCmd, 38
ballmerpeak::turtlenet::server::TurtlenetImpl
 addCategory, 43
 addComment, 43
 addKey, 44
 addMessageToCHAT, 44
 addPost, 44
 addToCategory, 44
 claimUsername, 45
 createCHAT, 45
 getCategories, 45
 getCategoryMembers, 45
 getComments, 45
 getConversation, 46
 getConversationMessages, 46
 getConversations, 46
 getConvoLastUpdated, 46
 getKey, 47
 getMyKey, 47
 getMyPDATA, 47
 getMyUsername, 47
 getPDATA, 47
 getPeople, 48
 getPostLastCommented, 48
 getUsername, 48
 getWallPosts, 48
 isFirstTime, 49
 like, 49
 register, 49
 removeFromCategory, 49
 revokeMyKey, 49
 sendPDATA, 50
 startTN, 50
 stopTN, 50
 timeMostRecentWallPost, 50
 unlike, 51
 updatePDATA, 51
 updatePDATAPermission, 51
Base64Decode
 ballmerpeak::turtlenet::server::Crypto, 4
Base64Encode
 ballmerpeak::turtlenet::server::Crypto, 4
calcRevocationKeys
 ballmerpeak::turtlenet::server::Database, 16
canSeePDATA
 ballmerpeak::turtlenet::server::Database, 16
claimName
 ballmerpeak::turtlenet::server::NetworkConnection,
 36
claimUsername
 ballmerpeak::turtlenet::server::TurtlenetImpl, 45
close
 ballmerpeak::turtlenet::server::NetworkConnection,
 37
createCHAT
 ballmerpeak::turtlenet::server::TurtlenetImpl, 45
createDBDir
 ballmerpeak::turtlenet::server::Database, 16
DBDirExists
 ballmerpeak::turtlenet::server::Database, 17
DBExists
 ballmerpeak::turtlenet::server::Database, 17
Database
 ballmerpeak::turtlenet::server::Database, 12
dbConnect
 ballmerpeak::turtlenet::server::Database, 16
dbCreate
 ballmerpeak::turtlenet::server::Database, 17
declineEvent
 ballmerpeak::turtlenet::server::Database, 17
decodeKey

ballmerpeak::turtlenet::server::Crypto, 4
decrypt
 ballmerpeak::turtlenet::server::Crypto, 4
decryptBytes
 ballmerpeak::turtlenet::server::Crypto, 5
decryptDB
 ballmerpeak::turtlenet::server::Crypto, 5
downloadNewMessages
 ballmerpeak::turtlenet::server::NetworkConnection,
 37

encodeKey
 ballmerpeak::turtlenet::server::Crypto, 5
encrypt
 ballmerpeak::turtlenet::server::Crypto, 5
encryptBytes
 ballmerpeak::turtlenet::server::Crypto, 6
encryptDB
 ballmerpeak::turtlenet::server::Crypto, 6
eraseContentFrom
 ballmerpeak::turtlenet::server::Database, 17
execute
 ballmerpeak::turtlenet::server::Database, 18

getAESKey
 ballmerpeak::turtlenet::server::Crypto, 6
getCategories
 ballmerpeak::turtlenet::server::Database, 18
 ballmerpeak::turtlenet::server::TurtlenetImpl, 45
getCategoryMembers
 ballmerpeak::turtlenet::server::Database, 18
 ballmerpeak::turtlenet::server::TurtlenetImpl, 45
getComments
 ballmerpeak::turtlenet::server::Database, 18
 ballmerpeak::turtlenet::server::TurtlenetImpl, 45
getConversation
 ballmerpeak::turtlenet::server::Database, 19
 ballmerpeak::turtlenet::server::TurtlenetImpl, 46
getConversationMessages
 ballmerpeak::turtlenet::server::Database, 19
 ballmerpeak::turtlenet::server::TurtlenetImpl, 46
getConversations
 ballmerpeak::turtlenet::server::Database, 19
 ballmerpeak::turtlenet::server::TurtlenetImpl, 46
getConvoLastUpdated
 ballmerpeak::turtlenet::server::TurtlenetImpl, 46
getKey
 ballmerpeak::turtlenet::server::Database, 19
 ballmerpeak::turtlenet::server::TurtlenetImpl, 47
getMessage
 ballmerpeak::turtlenet::server::NetworkConnection,
 37
getMyKey
 ballmerpeak::turtlenet::server::TurtlenetImpl, 47
getMyPDATA
 ballmerpeak::turtlenet::server::TurtlenetImpl, 47
getMyUsername
 ballmerpeak::turtlenet::server::TurtlenetImpl, 47
getName
 ballmerpeak::turtlenet::server::Database, 20
getPDATA
 ballmerpeak::turtlenet::server::Database, 20
 ballmerpeak::turtlenet::server::TurtlenetImpl, 47
getPeople
 ballmerpeak::turtlenet::server::TurtlenetImpl, 48
getPeopleInConvo
 ballmerpeak::turtlenet::server::Database, 20
getPostLastCommented
 ballmerpeak::turtlenet::server::TurtlenetImpl, 48
getPostVisibleTo
 ballmerpeak::turtlenet::server::Database, 20
getPrivateKey
 ballmerpeak::turtlenet::server::Crypto, 7
getPublicKey
 ballmerpeak::turtlenet::server::Crypto, 7
getSignatory
 ballmerpeak::turtlenet::server::Database, 21
getTestKey
 ballmerpeak::turtlenet::server::Crypto, 7
getTime
 ballmerpeak::turtlenet::server::NetworkConnection,
 37
getUsername
 ballmerpeak::turtlenet::server::TurtlenetImpl, 48
getVisibilityOfParent
 ballmerpeak::turtlenet::server::Database, 21
getWallPost
 ballmerpeak::turtlenet::server::Database, 21
getWallPostSender
 ballmerpeak::turtlenet::server::Database, 21
getWallPosts
 ballmerpeak::turtlenet::server::TurtlenetImpl, 48
hasMessage
 ballmerpeak::turtlenet::server::NetworkConnection,
 37
hash
 ballmerpeak::turtlenet::server::Crypto, 7
isFirstTime
 ballmerpeak::turtlenet::server::TurtlenetImpl, 49
isLiked
 ballmerpeak::turtlenet::server::Database, 22
isRevoked
 ballmerpeak::turtlenet::server::Database, 22
keyGen
 ballmerpeak::turtlenet::server::Crypto, 7
keysCanSeePDATA
 ballmerpeak::turtlenet::server::Database, 22
keysExist
 ballmerpeak::turtlenet::server::Crypto, 7
like
 ballmerpeak::turtlenet::server::Database, 22
 ballmerpeak::turtlenet::server::TurtlenetImpl, 49
NetworkConnection

ballmerpeak::turtlenet::server::NetworkConnection,
36
newADDCAT
 ballmerpeak::turtlenet::server::MessageFactory,
 30
newADDKEY
 ballmerpeak::turtlenet::server::MessageFactory,
 30
newADDTOCAT
 ballmerpeak::turtlenet::server::MessageFactory,
 31
newCHAT
 ballmerpeak::turtlenet::server::MessageFactory,
 31
newCLAIM
 ballmerpeak::turtlenet::server::MessageFactory,
 31
newCMNT
 ballmerpeak::turtlenet::server::MessageFactory,
 32
newEVNT
 ballmerpeak::turtlenet::server::MessageFactory,
 32
newLIKE
 ballmerpeak::turtlenet::server::MessageFactory,
 32
newMessage
 ballmerpeak::turtlenet::server::MessageFactory,
 33
newPCHAT
 ballmerpeak::turtlenet::server::MessageFactory,
 33
newPDATA
 ballmerpeak::turtlenet::server::MessageFactory,
 33
newPOST
 ballmerpeak::turtlenet::server::MessageFactory,
 34
newREMFROMCAT
 ballmerpeak::turtlenet::server::MessageFactory,
 34
newREVOKE
 ballmerpeak::turtlenet::server::MessageFactory,
 34
newUNLIKE
 ballmerpeak::turtlenet::server::MessageFactory,
 34
newUPDATECAT
 ballmerpeak::turtlenet::server::MessageFactory,
 35

postMessage
 ballmerpeak::turtlenet::server::NetworkConnection,
 37

query
 ballmerpeak::turtlenet::server::Database, 23

rand

Todo list

Bibliography

- [1] BBC. Azhar Ahmed convicted of offensive Facebook message. English. The BBC. Sept. 2012. URL: <http://www.bbc.co.uk/news/uk-england-leeds-19604735> (visited on 02/12/2014).
- [2] J. Callas et al. OpenPGP Message Format. English. RFC. Nov. 2007. URL: <https://tools.ietf.org/html/rfc4880> (visited on 02/11/2014).
- [3] Oracle Corporation. The Benefits of Risk Assessment on Projects, Portfolios, and Businesses. English. White Paper. Enterprise Software, Computer Software, June 2009. 14 pp. URL: <http://www.oracle.com/us/042743.pdf> (visited on 02/10/2014).
- [4] The Crown. Regulation of Investigatory Powers Act 2000. English. July 2000. URL: <http://www.legislation.gov.uk/ukpga/2000/23/section/50> (visited on 02/12/2014).
- [5] Wei Dai. Crypto++ 5.6.0 Benchmarks. English. Mar. 2009. URL: <http://www.cryptopp.com/benchmarks.html> (visited on 02/13/2014).
- [6] A. Dcosta and M. Gundlach. Stakeholders Risk Management - Project Risk Assessment Approach. English. Bright Hub Inc. Feb. 2014. URL: <http://www.brighthubpm.com/risk-management/33399-stakeholders-risk-management-project-risk-assessment-approach/> (visited on 02/10/2014).
- [7] Roger Dingledine. One cell is enough to break Tor's anonymity. English. Tor Project. Feb. 2009. URL: <https://blog.torproject.org/blog/one-cell-enough> (visited on 02/11/2014).
- [8] Facebook. Cookies, Pixels & Similar Technologies. English. 2014. URL: <https://www.facebook.com/help/cookies> (visited on 02/10/2014).
- [9] Facebook. Facebook Datause Policy. English. Nov. 2013. URL: <https://www.facebook.com/about/privacy> (visited on 02/10/2014).
- [10] Facebook. Statement of Rights and Responsibilities. English. Nov. 2013. URL: <https://www.facebook.com/legal/terms> (visited on 02/10/2014).

- [11] J. Scahill Glenn Greenwald. The NSAâŽs Secret Role in the U.S. Assassination Program. English. The Intercept. Feb. 2014. URL: <https://firstlook.org/theintercept/article/2014/02/10/the-nsas-secret-role/> (visited on 02/12/2014).
- [12] K.L. Huff. WebPG for Mozilla. English. Jan. 2013. URL: <https://addons.mozilla.org/en-US/firefox/addon/webpg-firefox/> (visited on 02/13/2014).
- [13] Oracle Inc. Java. Features and Benefits. English. Oracle Inc. Feb. 2014. URL: <http://www.oracle.com/us/technologies/java/features/index.html> (visited on 02/10/2014).
- [14] Prosci Inc. Change Management: The Systems and Tools for Managing Change. English. Prosci Inc. Feb. 2014. URL: <http://www.change-management.com/tutorial-change-process-detailed.htm#Definition> (visited on 02/10/2014).
- [15] The Tor Project Inc. Tor Project: Overview. English. Feb. 2014. URL: <https://www.torproject.org/about/overview.html.en> (visited on 02/13/2014).
- [16] J. Garside J. Ball L. Harding. BT and Vodafone among telecoms companies passing details to GCHQ. English. The Guardian. Aug. 2013. URL: <http://www.theguardian.com/business/2013/aug/02/telecoms-bt-vodafone-cables-gchq> (visited on 02/12/2014).
- [17] J.P. Lewis and U. Neumann. Performance of Java versus C++. English. Comparison. California, USA: Computer Graphics and Immersive Technology Lab - University of Southern California, 2004. URL: <http://scribblethink.org/Computer/javaCbenchmark.html> (visited on 02/10/2014).
- [18] Known Bad Relays. English. Tor Project. Jan. 2014. URL: <https://trac.torproject.org/projects/tor/wiki/doc/badRelays> (visited on 02/11/2014).
- [19] C. Labovitz. Libya Firewall Begins to Crumble? English. Feb. 2011. URL: <http://www.monkey.org/~labovit/blog/> (visited on 02/27/2011).
- [20] Peter Maass. How Laura Poitras Helped Snowden Spill His Secrets. English. Aug. 2013. URL: <http://www.nytimes.com/2013/08/18/magazine/laura-poitras-snowden.html?smid=pl-share> (visited on 02/10/2014).
- [21] Anna Mar. 130 Project Risks (List). English. Mar. 2013. URL: <http://management.simplifiable.com/management/new/130-project-risks> (visited on 02/10/2014).
- [22] Pinsent Masons. Law requiring disclosure of decryption keys in force. English. Pinsent Masons LLP. Oct. 2007. URL: <http://www.out-law.com/page-8515> (visited on 02/10/2014).
- [23] General Public. x86-64. Differences between AMD64 and Intel 64. English. Wikimedia Foundation Inc. Feb. 2014. URL: http://en.wikipedia.org/wiki/X86-64#Differences_between_AMD64_and_Intel_64 (visited on 02/10/2014).

- [24] J. Risen. N.S.A. Gathers Data on Social Connections of U.S. Citizens. English. The New York Times. Sept. 2013. URL: http://www.nytimes.com/2013/09/29/us/nsa-examines-social-networks-of-us-citizens.html?_r=1&pagewanted=all& (visited on 02/12/2014).
- [25] Tom Ritter. “De-Anonymizing Alt.Anonymous.Messages”. In: Defcon 21, Nov. 2013. URL: https://www.youtube.com/watch?v=_Tj6c2Ikq_E (visited on 02/11/2014).
- [26] Tom Ritter. The Differences Between Onion Routing and Mix Networks. English. Apr. 2013. URL: http://ritter.vg/blog-mix_and_onion_networks.html (visited on 02/13/2014).
- [27] P. Rose. UK Court Parts with US Court regarding Compelled Disclosure of Encryption Keys. English. Proskauer Rose LLP. Oct. 2008. URL: <http://privacylaw.proskauer.com/2008/10/articles/international/uk-court-parts-with-us-court-regarding-compelled-disclosure-of-encryption-keys/> (visited on 02/10/2014).
- [28] Richard M. Smith. The Web Bug FAQ. English. Nov. 1999. URL: <http://w2.eff.org/Privacy/Marketing/web\bug.html> (visited on 02/10/2014).
- [29] J. Travers and S. Milgram. “An Experimental Study of the Small World Problem”. English. In: Sociometry 32.4 (Dec. 1969). URL: http://www.cis.upenn.edu/~mkearns/teaching/NetworkedLife/travers_milgram.pdf (visited on 02/11/2014).
- [30] unknown. Tor Stinks. English. NSA. June 2012. URL: <http://media.encrypted.cc/files/nsa/tor-stinks.pdf> (visited on 02/11/2014).
- [31] various. Global surveillance disclosures (2013â€¢present). English. Wikipedia. Feb. 2014. URL: https://en.wikipedia.org/wiki/Global_surveillance_disclosures_%282013%E2%80%93present%29 (visited on 02/12/2014).
- [32] P. Wiseman. Cracking the 'Great Firewall' of China's Web censorship. English. USA TODAY. Apr. 2008. URL: <http://abcnews.go.com/Technology/story?id=4707107> (visited on 04/24/2008).