OTR Messenger

Ian Martiny, Sergey Frolov, Shirly Montero Quesada

Table 1. Features implemented

Requirement	<user>	<task>	<accomplish my goal>	priority
UR-001	client	sign up	to create an account and access chat service	High
UR-002	client	log in	to access the chat service	High
UR-003	client	send messages	to communicate with others	Critical
UR-004	client	receive messages	to communicate with others	Critical
UR-005	client	view friends list	to see who is online	High
UR-006	client	modify friends list	to update the list	Medium
UR-010	admin	change server status (launch/reset/terminate)	provide, temporarily stop or terminate service	Critical
UR-011	admin	view/log list of all users	part of documentation to monitor the system	High
UR-012	admin	view logged in users	manage the system	Medium
UR-013	admin	view keys (have access the database)	security analysis on keys (not repeating)	Medium

Functional Requirement	Area: specifications
FNFR-001	Security: On account creation a public and a private key will be generated
FNFR-002	Security: Create a shared encryption key as two users connect with each other (exchange each others public key)
FNFR-006	Performance: after log in it takes 7 s to show friend's list
FNFR-007	Performance: 2 s after showing friends list window, show list of friends who are online

Table 2. Features not implemented

Requirements	Specifications	Topic Area	Actor	Priority
BR-001	Password at least 8 characters and consists of, at least, and one uppercase, one lowercase char, one special char, and one number	Sign up	All	Medium

Functional Requirement	Area: specifications
FNFR-003	Security: After some time out 60s, the users will publish their private signing keys and new ones will be generated.
FNFR-004	Legal: Once a new shared/private signing key is generated, the user will be alerted of the change
FNFR-005	Legal: If user wants to override a key change time setting, an advice will be generated (if longer time then alert of the consequences)

Requirement	<user>	<task>	<accomplish my goal>	priority
UR-007	client	organize friends list	to modify groups in the list	High
UR-008	client	view keys (public/private)	so I can verify them	Medium
UR-009	client	request change key	to communicate securely	Medium
UR-014	client	manually change key	to have direct control over my security	Nice-to-have
UR-015	client	access password	to view it, modify it	Low
UR-016	client	import contact list (select/deselect)	to add many friends at once	Nice-to-have
UR-017	client	upload/download files	to send/receive more data	Low
UR-018	client	read old messages (memento)	to review what was said	Low
UR-019	client	reject/accept invitations to be added to a list	to decide who I want to talk with	Nice-to-have
UR-020	client	black lists other clients	to block others from bothering me	Low

Class Diagrams: what changed and why

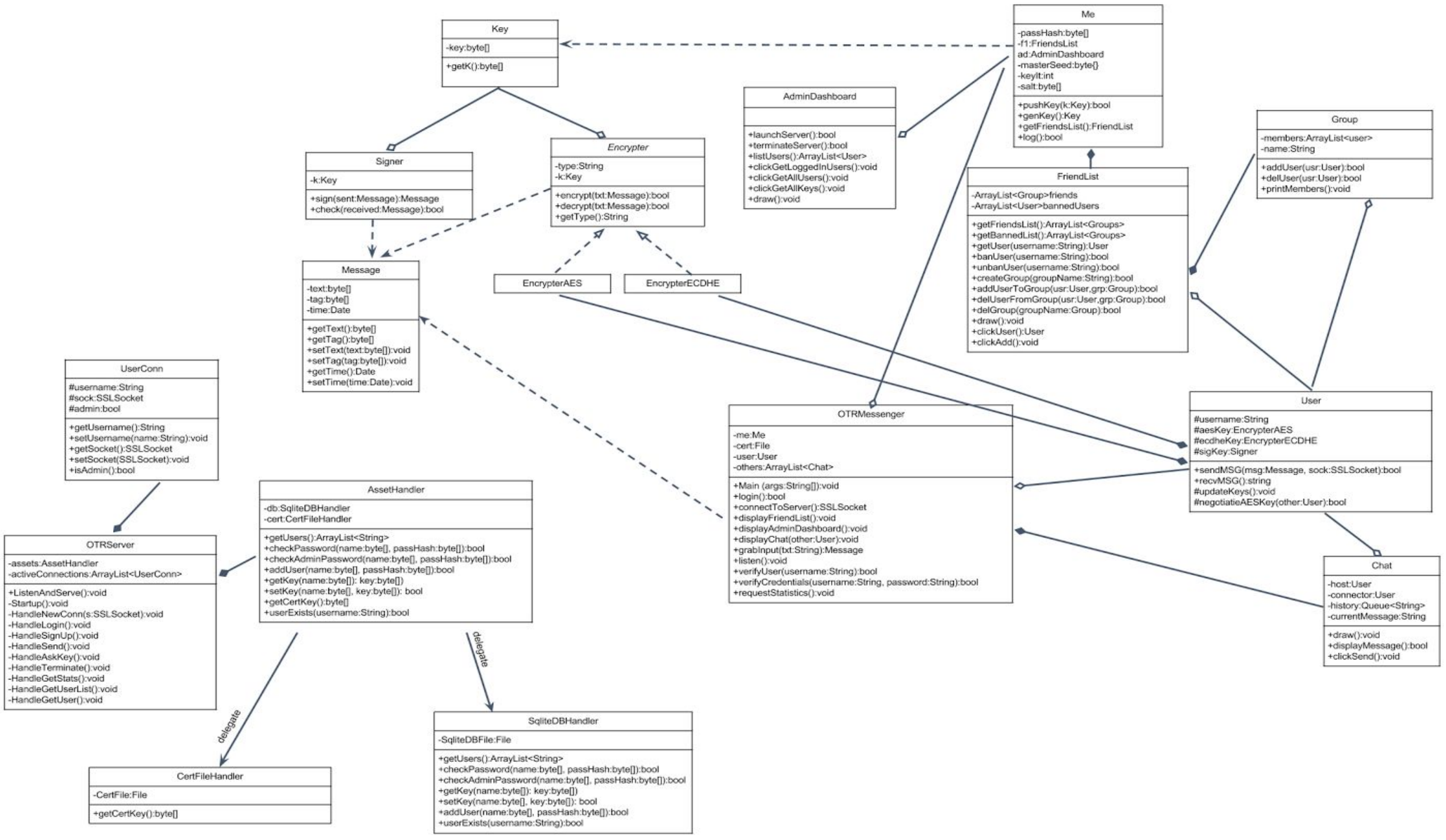
Key
-key:byte[]
+getK():byte[]
Me
-passHash:byte[]
-f1:FriendsList
ad:AdminDashboard
-masterSeed:byte{}
-keyIt:int
-salt:byte[]
+pushKey(k:Key):bool
+genKey():Key
+getFriendsList():FriendList
+log():bool
AdminDashboard
+launchServer():bool
+terminateServer():bool
+listUsers():ArrayList<User>
+clickGetLoggedInUsers():void
+clickGetAllUsers():void
+clickGetAllKeys():void
+draw():void
Group
-members:ArrayList<user>
-name:String
+addUser(usr:User):bool
+delUser(usr:User):bool
+printMembers():void
Signer
-k:Key
+sign(sent:Message):Message
+check(received:Message):bool
Encrypter
-type:String
-k:Key
+encrypt(txt:Message):bool
+decrypt(txt:Message):bool
+getType():String
FriendList
-ArrayList<Group>friends
-ArrayList<User>bannedUsers
+getFriendsList():ArrayList<Groups>
+getBannedList():ArrayList<Groups>
+getUser(username:String):User
+banUser(username:String):bool
+unbanUser(username:String):bool
+createGroup(groupName:String):bool
+addUserToGroup(usr:User,grp:Group):bool
+delUserFromGroup(usr:User,grp:Group):bool
+delGroup(groupName:Group):bool
+draw():void
+clickUser():User
+clickAdd():void
Message
-text:byte[]
-tag:byte[]
-time:Date
+getText():byte[]
+getTag():byte[]
+setText(text:byte[]):void
+setTag(tag:byte[]):void
+getTime():Date
+setTime(time:Date):void
EncrypterAES
EncrypterECDHE
UserConn
#username:String
#sock:SSLSocket
#admin:bool
+getUsername():String
+setUsername(name:String):void
+getSocket():SSLSocket
+setSocket(SSLSocket):void
+isAdmin():bool
User
#username:String
#aesKey:EncrypterAES
#ecdheKey:EncrypterECDHE
#sigKey:Signer
+sendMSG(msg:Message, sock:SSLSocket):bool
+recvMSG():string
#updateKeys():void
#negotiatieAESKey(other:User):bool
OTRMessenger
-me:Me
-cert:File
-user:User
-others:ArrayList<Chat>
+Main (args:String[]):void
+login():bool
+connectToServer():SSLSocket
+displayFriendList():void
+displayAdminDashboard():void
+displayChat(other:User):void
+grabInput(txt:String):Message
+listen():void
+verifyUser(username:String):bool
+verifyCredentials(username:String, password:String):bool
+requestStatistics():void
AssetHandler
-db:SqliteDBHandler
-cert:CertFileHandler
+getUsers():ArrayList<String>
+checkPassword(name:byte[], passHash:byte[]):bool
+checkAdminPassword(name:byte[], passHash:byte[]):bool
+addUser(name:byte[], passHash:byte[]):bool
+getKey(name:byte[]): key:byte[])
+setKey(name:byte[], key:byte[]): bool
+getCertKey():byte[]
+userExists(username:String):bool
OTRServer
-assets:AssetHandler
-activeConnections:ArrayList<UserConn>
+ListenAndServe():void
-Startup():void
-HandleNewConn(s:SSLSocket):void
-HandleLogin():void
-HandleSignUp():void
-HandleSend():void
-HandleAskKey():void
-HandleTerminate():void
-HandleGetStats():void
-HandleGetUserList():void
-HandleGetUser():void
Chat
-host:User
-connector:User
-history:Queue<String>
-currentMessage:String
+draw():void
+displayMessage():bool
+clickSend():void
delegate
delegate
SqliteDBHandler
-SqliteDBFile:File
+getUsers():ArrayList<String>
+checkPassword(name:byte[], passHash:byte[]):bool
+checkAdminPassword(name:byte[], passHash:byte[]):bool
+getKey(name:byte[]): key:byte[])
+setKey(name:byte[], key:byte[]): bool
+addUser(name:byte[], passHash:byte[]):bool
+userExists(username:String):bool
CertFileHandler
-CertFile:File
+getCertKey():byte[]

Figure 1. Original class diagram (see original in folder support materials)

We made a conceptual server-side change and moved responsibility to process the requests from OTRServer class to UserConn: functions like HandleLogin() and HandleSend() are now in UserConn. We concluded that it is reasonable for UserConn class to be processing requests happening in this connection, OTRServer now is simply spins up the server, and all user connections are immediately delegated to UserConn. In addition, UserConn now *implements Runnable*, thus is independently ran in a separate thread, further decreasing communication and coupling with OTRServer.

Additionally, we made many reconfigurations on the client side. One of the issues that we noticed when implementing (and discussing) the project was how the objects communicated information to each other. Thus during our refactoring we focused on how we wanted objects to communicate with each other. To facilitate this we implemented a ServerConnector class to work as the sole communicator between server and client. Additionally we created a Host class that would be passed around as the currently logged in user.

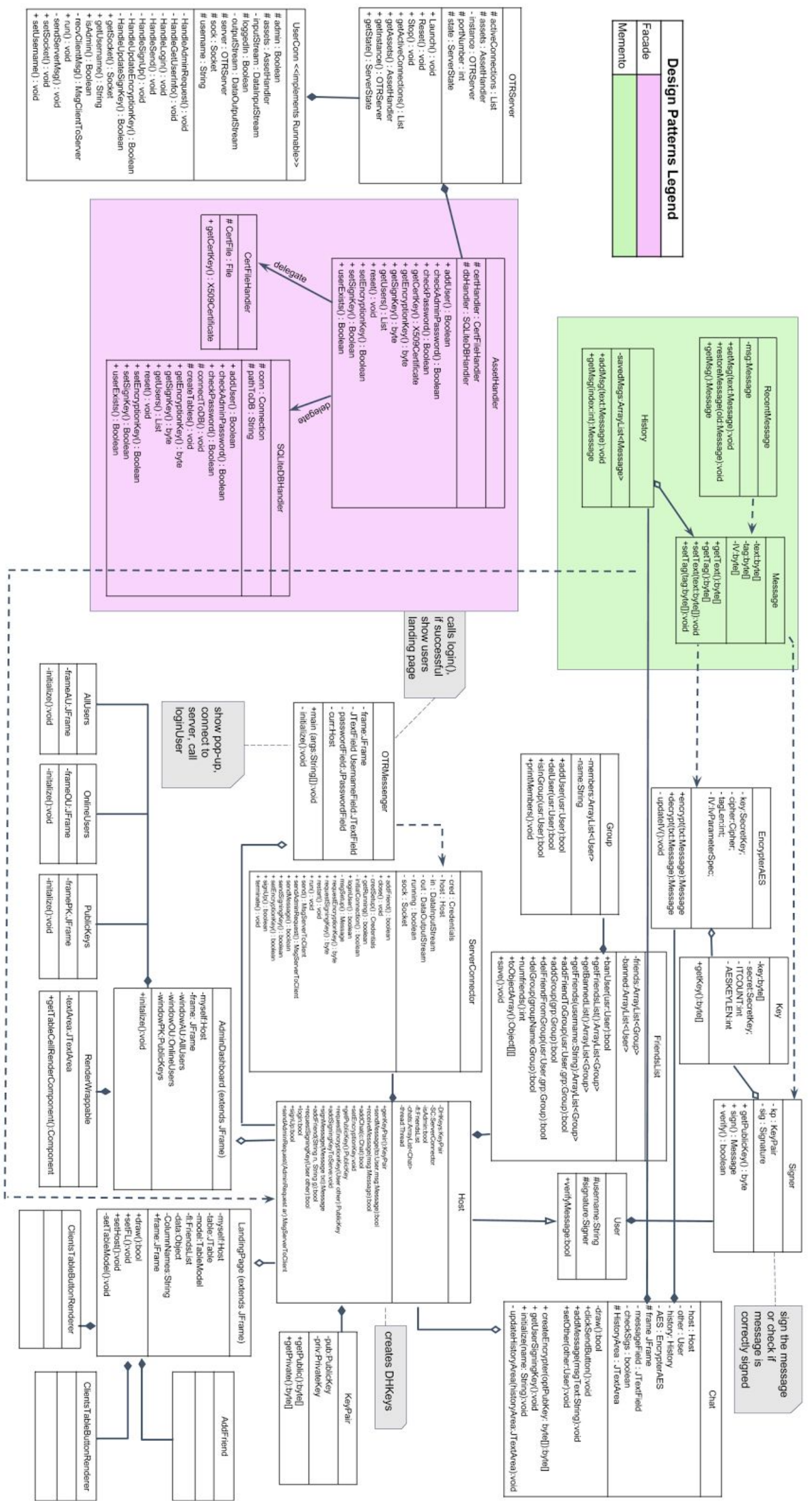

Figure 2. New class diagram (see original in folder support materials)

Design Patterns used

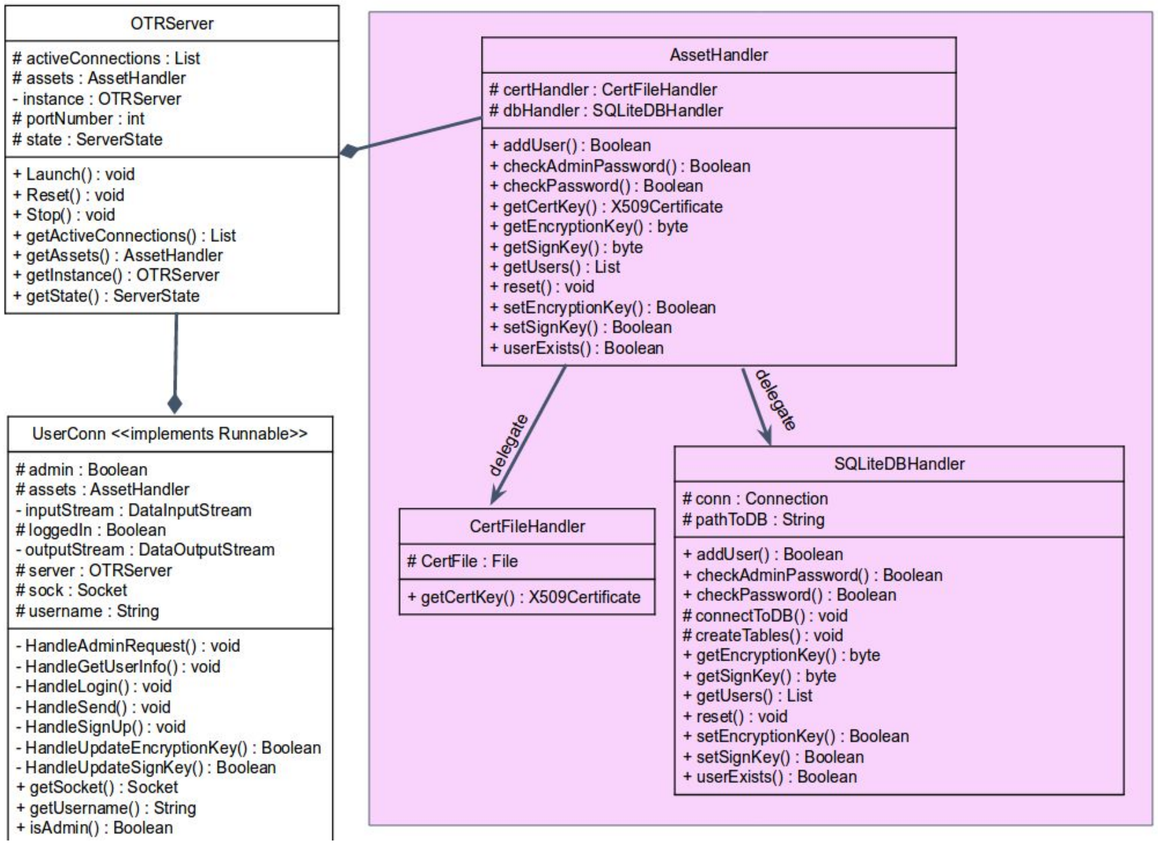

Figure 3: Facade pattern used

The server side uses **Facade** for assets handling (see our final class diagram on Figure 3). AssetHandler has functions for getting/saving user info and x509 certificates, which are delegated to various managers. This way we are providing a level of abstraction that can easily change the form in which any subset data is stored and change the database, such that we won't have to change code anywhere, but in those handlers.

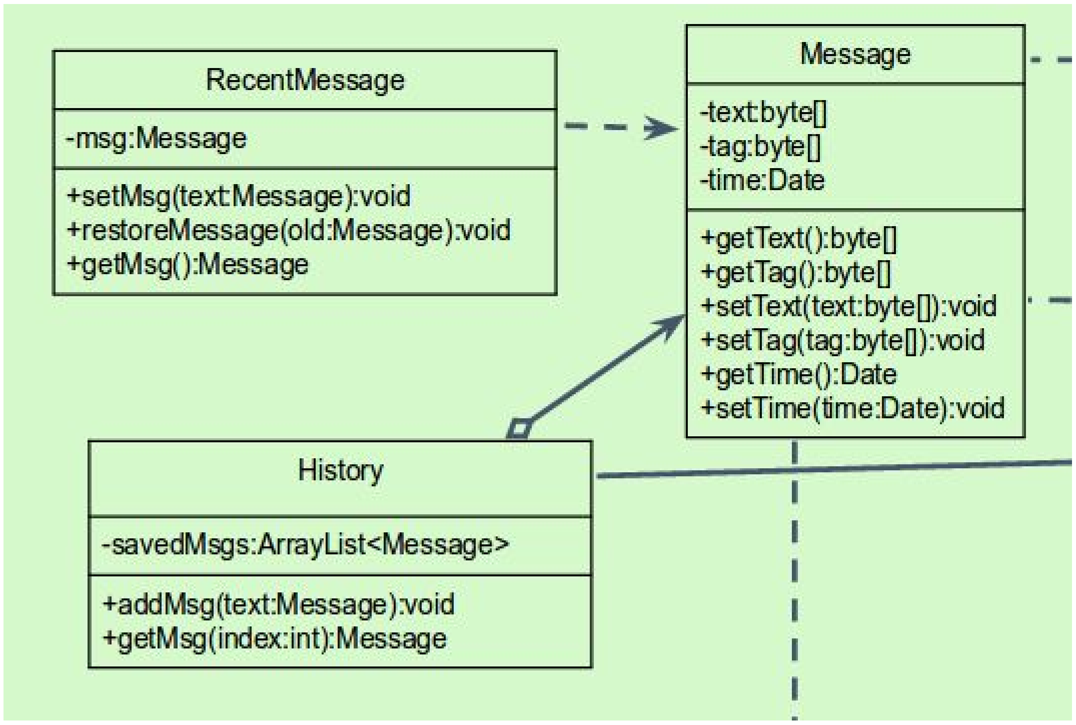

Figure 4: Memento class diagram

We intended to implement **Memento** on the client side to keep track of the history of communications between clients. The Message class keeps track of particular messages (as the Memento) and the History class keeps track of all messages between two clients (as the Caretaker). However due to time constraints we were unable to completely finish this. Though we did make use of the History and Message class for viewing/sending messages.

Lessons learned

- This class helped us learn how to step back from just coding and focus on how to design our project in such a way that we all had a clear idea of how things worked. Indeed, design is an important and iterative process, which continues throughout the development process, and it's important to keep working on both design and implementation.
- Clear and readable logging makes debugging significantly easier.
- The importance of starting over from "scratch" when necessary to get a clear idea of how our project parts work together. We learned how to step back from just coding and focus on how to design our project in such a way that we all had a clear idea of how things worked.
- Additionally, we gained experience with tools, such as Eclipse/Intellij and how to handle 3-rd party libraries (e.g. protobuf and mysqlite3) and how to work in a team using GIT. We also realized that the list of important tools includes not only IDEs and git, but also UML diagram generators and Google Docs, that go long way in making software engineer's life easier.