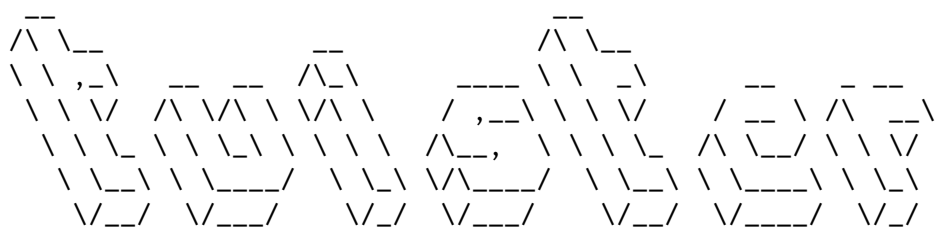


Object-Oriented Programming Techniques

Project 3 - SocialApp

Juan Antonio Aldea Armenteros - 0404450
Pablo Caro Martin - 0404489

Project Description

In this project, a Social Network App has been implemented, presenting the following functionalities:

- The server is able to support multiple simultaneous clients.
- New users are able to register to the system with user name and password.
- Registered users are able to login into the system.
- After logging into the system, an user can:
  - Update his/her own status by sending a message to the server. Server saves the timestamp of the status update.
  - Get the list of all registered users.
  - Start "following" a specific user.
  - Stop “following” a specific user.
  - Get the list of accounts the user is currently following.
  - Get the feed of all followed users (including the user itself) and their status messages.
  - "Like" a selected status update.
  - “Unlike” a previously “liked” status update.
  - Share content. The user's share page is available to every user, even those who are not logged into the system.

- Model-View-Controller design pattern used.
- Server stores all the data in a SQLite3 database.

Instructions for use

This software implements server and client capabilities. In order to chose the function mode, the application has to be executed with the following parameters:

Server mode:

SocialApp 0 <port> -> Open the server, listening on the specified port.

Client mode:

SocialApp 1 <host IP> <port> ->

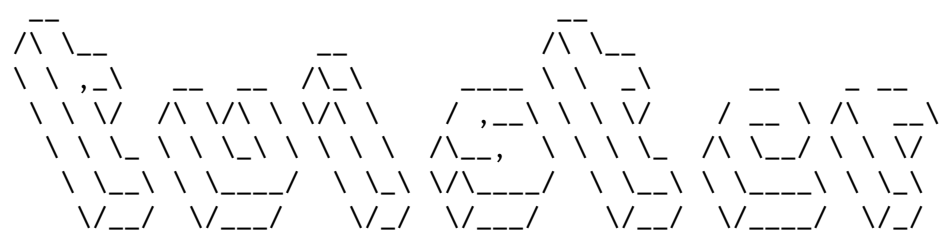
When the client is not logged in, only the following commands are available:

- **register** <*username*> <*password*> -> Adds a new user in the system.
- **login** <*username*> <*password*> -> Access with an existing username and password.
- **usercontent** <*username*> -> Gets the content shared by the specified user. Can be done when not logged in.

When the client is logged in, the following commands are available:

- **logout** -> Closes the session.
- **update** -> Gets the last posts of the people you are following.
- **publish** <*text*> -> Shares some content with your followers.
- **follow** <*username*> -> Subscribes to the content of an user.
- **unfollow** <*username*> -> Unsubscribes to the content of an user.
- **like** <*id*> -> Marks the comment as something you like.
- **unlike** <*id*> -> Unmarks the comment as something you like.
- **following** -> Lists the users you are currently following.
- **usercontent** <*username*> -> Gets the content shared by the specified user. Can be done when not logged in.
- **listusers** -> Lists the users registered in the system.

Session example



```
Welcome to the Tuister client. Enter "help" for detailed instructions of
available commands

login Pablo password
Sending login request...
Logged in.

update
[3] Pablo - 2013-04-21 10:20:13 - 1 like
    This is the new thing.

[2] Luisfe - 2013-04-21 14:19:39 - 0 likes
    I don't understand it well...

[1] Elena - 2013-04-21 14:20:47 - 2 likes
    As we say in Spain, "está chupao".


like 2
Done.

listusers
- Pablo
- Juan
- Elena
- Luisfe
- Jussi

follow Juan
Done.

update
[4] Pablo - 2013-04-21 10:20:13 - 1 like
    This is the new thing.

[3] Luisfe - 2013-04-21 14:19:39 - 0 likes
    I don't understand it well...

[2] Elena - 2013-04-21 14:20:47 - 2 likes
    As we say in Spain, "está chupao".

[1] Juan - 2013-04-21 14:20:47 - 0 likes
    I still prefer Twitter.
```

```
publish Twitter is dead, Tuister is the real deal.
Done.

update
[5] Pablo - 2013-04-21 10:20:13 - 1 like
    This is the new thing.

[4] Luisfe - 2013-04-21 14:19:39 - 0 likes
    I don't understand it well...

[3] Elena - 2013-04-21 14:20:47 - 2 likes
    As we say in Spain, "está chupao".

[2] Juan - 2013-04-21 14:21:49 - 0 likes
    I still prefer Twitter.

[1] Pablo - 2013-04-21 14:32:20 - 0 likes
    Twitter is dead, Tuister is the real deal.


usercontent Jussi
[2] Jussi - 2013-04-10 11:51:11 - 12 likes
    I was using Tuister before it was cool.

[1] Jussi - 2013-04-11 12:32:20 - 15 likes
    I like Linux because of the penguin.


logout
Done.
Logged out.

exit
Exitting

[End of the execution]
```

Design patterns

In the design and implementation of this project, the following patterns have been applied:

- **Model-View-Controller**: The base of the client functionality. The tasks are divided as follows:

    - Controller: The controller handles the general behaviour of the client application, using a State Pattern. It updates the View when needed, and handles the networking. Depends on the View to interact with the user, and on the Model to parse the network inputs.

    - View: In this case, although there is not a real graphical user interface, the console takes the role of view. It interacts directly with the user. When the user writes a command, the command itself and its parameters are filtered, and the corresponding method in the controller is called. It is updated by the Controller.

    - Model: It handles the data parsing, forming structures for the Controller to use them. In the case of posts, it handles the mapping between the local ID (the ID shown in the list of the client) and the global ID (the ID given by the server).

- **State**: Used in both the Client and the Server, this pattern helps when defining the behaviour of the application, basing it on a State Machine. The State Handler object has defined every method which depend on the current state to function, and delegates it on the Current State; modifying the current state if needed. This way, the Client and the Server themselves do not need to worry about the state, as the State Handler take responsibility for them.

- **Singleton**: This pattern is applied in the database wrapper and the server state. In the case of the database wrapper, the Singleton Pattern guarantees the correct function of the database connection, when more than one thread have access to it. As the response of the concurrent access could depend on the driver, this is the best way to establish a fixed behaviour.

Reusability and Extensibility

In this project, the reusability and extensibility of the software have been two of the main concerns in the design phase. For this reason, some measures were taken:

- The use of the Model-View-Controller pattern guarantees the reusability of each one of the conforming parts. For example, adding a graphical user interface to the client part of the application would be extremely easy, because of the correct integration between the View and the Controller.

- The use of State Patterns in the client and the server makes very easy to define new behaviours, adding them to the corresponding state, without altering the general structure.

- Concerning the serialization and deserialization of the data in order to prepare a message to send it by the network, XML has been used. In this case, JACB (Java Architecture for XML Binding) allows adding new messages to the protocol, as well as new content for existing messages, extremely easy.

- Also, with XML, a parser SAX can be used to control the behaviour, with the only work of assigning callbacks to the different tags.

- Again, as the logic and the model are separated, it would be easy to change the storage system used by the server. The database wrapper acts like an interface to the application, isolating the database management system to the rest.

Division of responsibilities

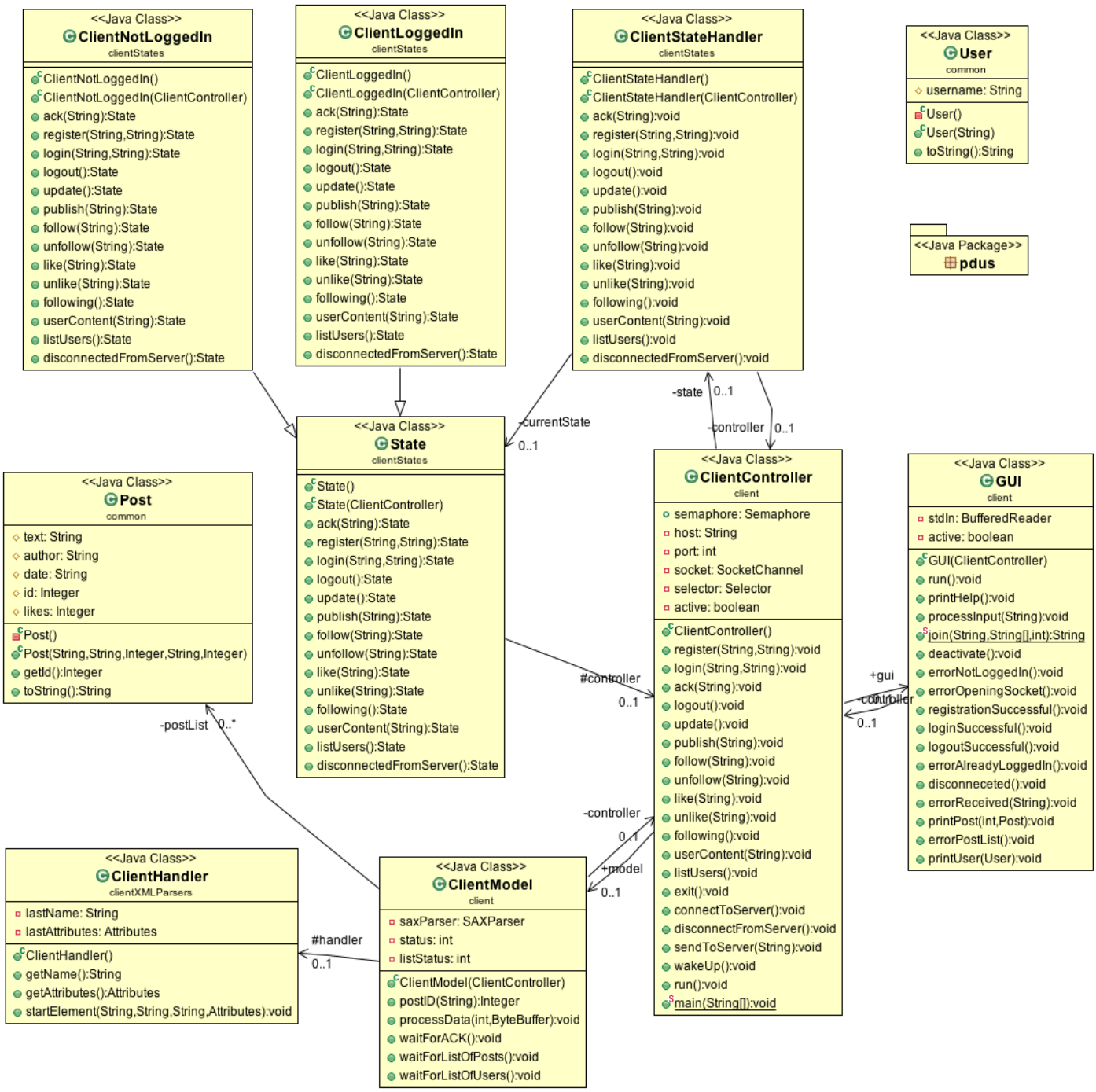
Juan Antonio Aldea Armenteros - 0404450

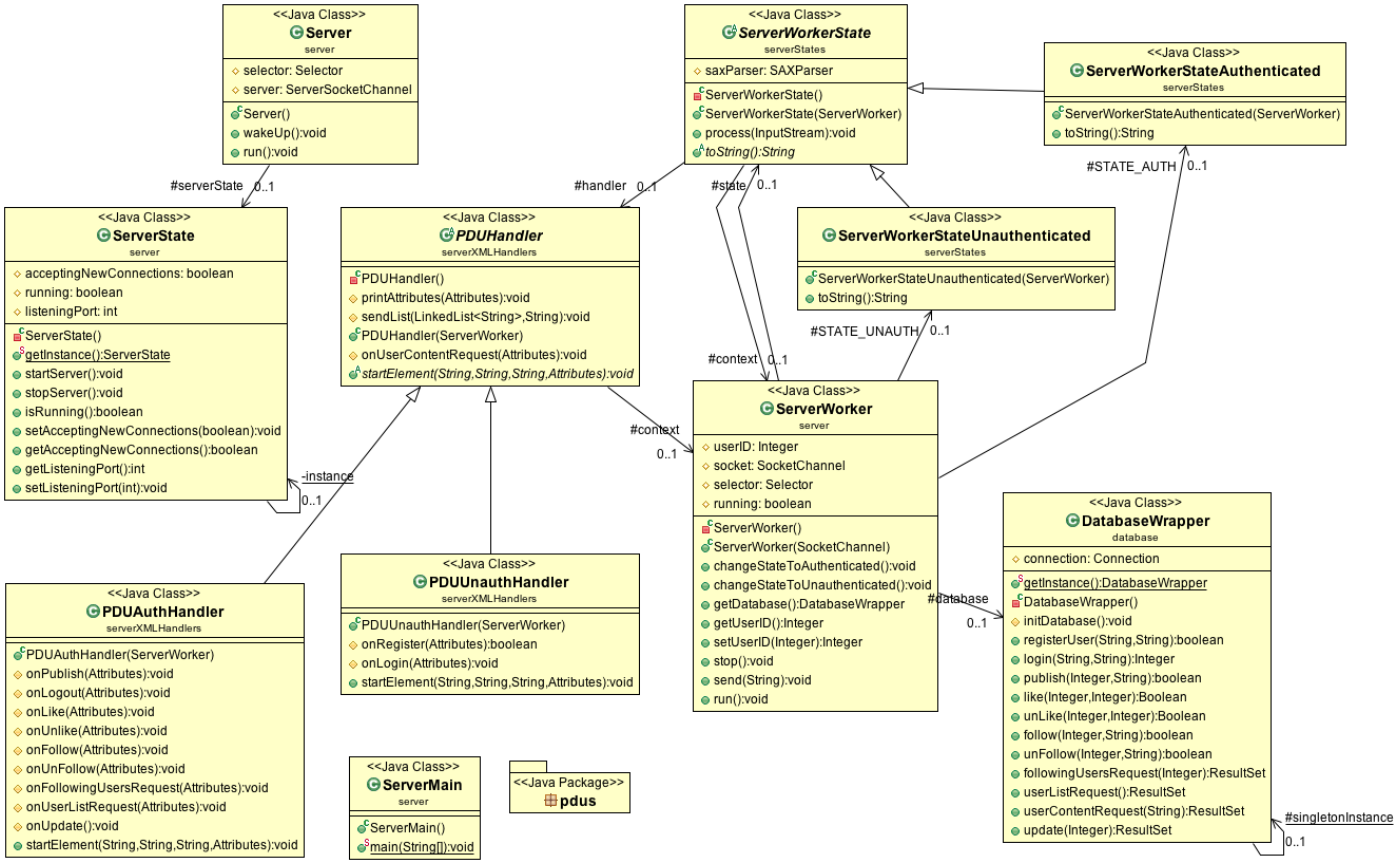
- Server part of the application.
- Database handling and integration.
- XML generation and parsing.

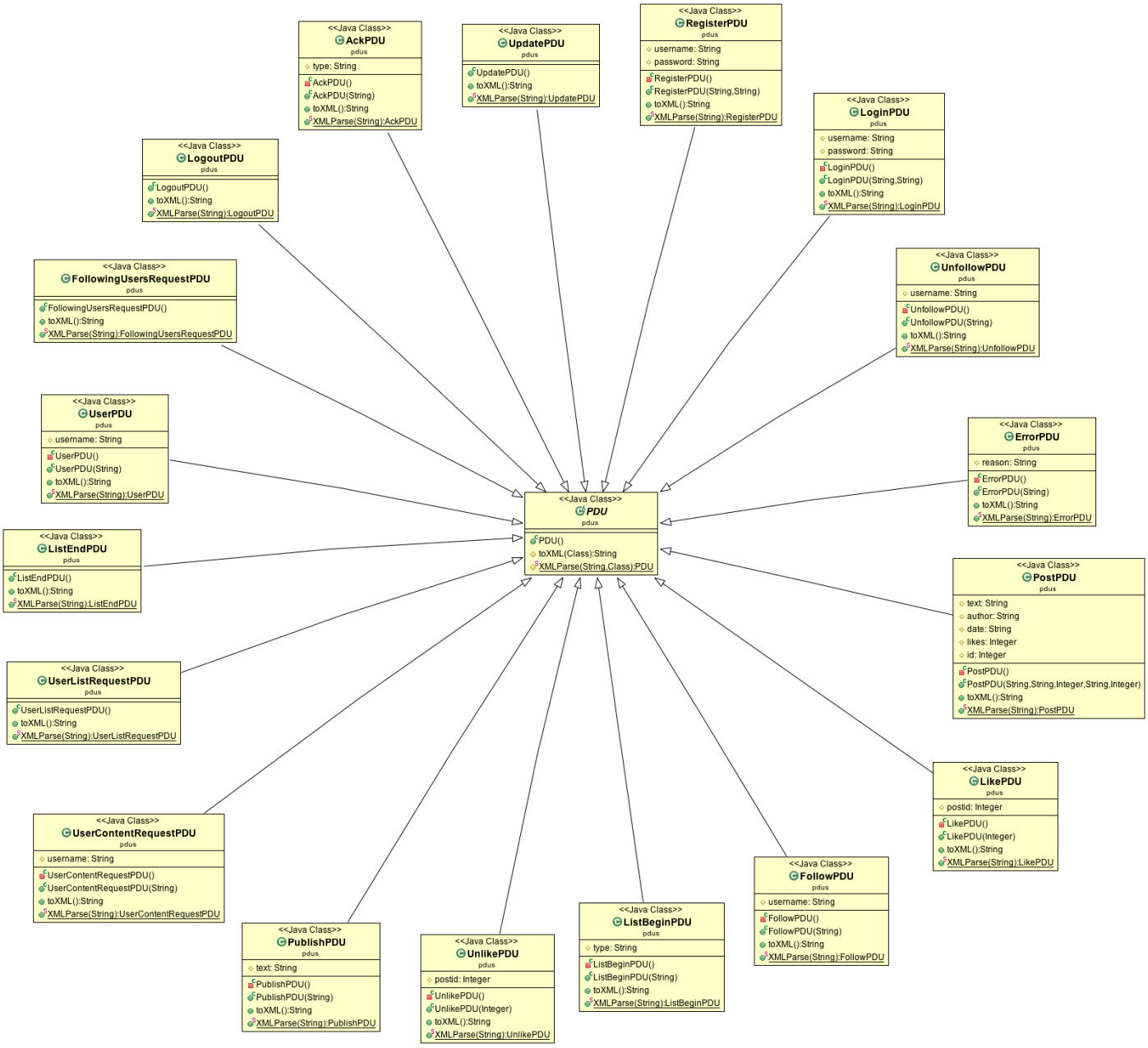
Pablo Caro Martin - 0404489

- Client part of the application.
- Project report.

Appendix: Class Diagrams


<<Java Class>>
ClientNotLoggedIn
clientStates
ClientNotLoggedIn()
ClientNotLoggedIn(ClientController)
ack(String):State
register(String,String):State
login(String,String):State
logout():State
update():State
publish(String):State
follow(String):State
unfollow(String):State
like(String):State
unlike(String):State
following():State
userContent(String):State
listUsers():State
disconnectedFromServer():State
<<Java Class>>
ClientLoggedIn
clientStates
ClientLoggedIn()
ClientLoggedIn(ClientController)
ack(String):State
register(String,String):State
login(String,String):State
logout():State
update():State
publish(String):State
follow(String):State
unfollow(String):State
like(String):State
unlike(String):State
following():State
userContent(String):State
listUsers():State
disconnectedFromServer():State
<<Java Class>>
ClientStateHandler
clientStates
ClientStateHandler()
ClientStateHandler(ClientController)
ack(String):void
register(String,String):void
login(String,String):void
logout():void
update():void
publish(String):void
follow(String):void
unfollow(String):void
like(String):void
unlike(String):void
following():void
userContent(String):void
listUsers():void
disconnectedFromServer():void
<<Java Class>>
User
common
username: String
User()
User(String)
toString():String
<<Java Package>>
pdus
<<Java Class>>
State
clientStates
State()
State(ClientController)
ack(String):State
register(String,String):State
login(String,String):State
logout():State
update():State
publish(String):State
follow(String):State
unfollow(String):State
like(String):State
unlike(String):State
following():State
userContent(String):State
listUsers():State
disconnectedFromServer():State
-currentState
0..1
-state
0..1
-controller
0..1
<<Java Class>>
Post
common
text: String
author: String
date: String
id: Integer
likes: Integer
Post()
Post(String,String,Integer,String,Integer)
getId():Integer
toString():String
-postList
0..*
<<Java Class>>
ClientController
client
semaphore: Semaphore
host: String
port: int
socket: SocketChannel
selector: Selector
active: boolean
ClientController()
register(String,String):void
login(String,String):void
ack(String):void
logout():void
update():void
publish(String):void
follow(String):void
unfollow(String):void
like(String):void
unlike(String):void
following():void
userContent(String):void
listUsers():void
exit():void
connectToServer():void
disconnectFromServer():void
sendToServer(String):void
wakeUp():void
run():void
main(String[]):void
#controller
0..1
+gui
-controller
0..1
<<Java Class>>
GUI
client
stdIn: BufferedReader
active: boolean
GUI(ClientController)
run():void
printHelp():void
processInput(String):void
join(String,String[],int):String
deactivate():void
errorNotLoggedIn():void
errorOpeningSocket():void
registrationSuccessful():void
loginSuccessful():void
logoutSuccessful():void
errorAlreadyLoggedIn():void
disconneceted():void
errorReceived(String):void
printPost(int,Post):void
errorPostList():void
printUser(User):void
-controller
0..1
+model
0..1
<<Java Class>>
ClientHandler
clientXMLParsers
lastName: String
lastAttributes: Attributes
ClientHandler()
getName():String
getAttributes():Attributes
startElement(String,String,String,Attributes):void
#handler
0..1
<<Java Class>>
ClientModel
client
saxParser: SAXParser
status: int
listStatus: int
ClientModel(ClientController)
postID(String):Integer
processData(int,ByteBuffer):void
waitForACK():void
waitForListOfPosts():void
waitForListOfUsers():void


<<Java Class>>
Server
server
selector: Selector
server: ServerSocketChannel
Server()
wakeUp():void
run():void
<<Java Class>>
ServerWorkerState
serverStates
saxParser: SAXParser
ServerWorkerState()
ServerWorkerState(ServerWorker)
process(InputStream):void
toString():String
<<Java Class>>
ServerWorkerStateAuthenticated
serverStates
ServerWorkerStateAuthenticated(ServerWorker)
toString():String
#serverState 0..1
#handler 0..1
#state 0..1
#STATE_AUTH 0..1
<<Java Class>>
ServerState
server
acceptingNewConnections: boolean
running: boolean
listeningPort: int
ServerState()
getInstance():ServerState
startServer():void
stopServer():void
isRunning():boolean
setAcceptingNewConnections(boolean):void
getAcceptingNewConnections():boolean
getListeningPort():int
setListeningPort(int):void
<<Java Class>>
PDUHandler
serverXMLHandlers
PDUHandler()
printAttributes(Attributes):void
sendList(LinkedList<String>,String):void
PDUHandler(ServerWorker)
onUserContentRequest(Attributes):void
startElement(String,String,String,Attributes):void
<<Java Class>>
ServerWorkerStateUnauthenticated
serverStates
ServerWorkerStateUnauthenticated(ServerWorker)
toString():String
#STATE_UNAUTH 0..1
#context 0..1
-instance
0..1
#context
0..1
<<Java Class>>
ServerWorker
server
userID: Integer
socket: SocketChannel
selector: Selector
running: boolean
ServerWorker()
ServerWorker(SocketChannel)
changeStateToAuthenticated():void
changeStateToUnauthenticated():void
getDatabase():DatabaseWrapper
getUserID():Integer
setUserID(Integer):Integer
stop():void
send(String):void
run():void
<<Java Class>>
DatabaseWrapper
database
connection: Connection
getInstance():DatabaseWrapper
DatabaseWrapper()
initDatabase():void
registerUser(String,String):boolean
login(String,String):Integer
publish(Integer,String):boolean
like(Integer,Integer):Boolean
unLike(Integer,Integer):Boolean
follow(Integer,String):boolean
unFollow(Integer,String):boolean
followingUsersRequest(Integer):ResultSet
userListRequest():ResultSet
userContentRequest(String):ResultSet
update(Integer):ResultSet
#database
0..1
#singletonInstance
0..1
<<Java Class>>
PDUAuthHandler
serverXMLHandlers
PDUAuthHandler(ServerWorker)
onPublish(Attributes):void
onLogout(Attributes):void
onLike(Attributes):void
onUnlike(Attributes):void
onFollow(Attributes):void
onUnFollow(Attributes):void
onFollowingUsersRequest(Attributes):void
onUserListRequest(Attributes):void
onUpdate():void
startElement(String,String,String,Attributes):void
<<Java Class>>
PDUUnauthHandler
serverXMLHandlers
PDUUnauthHandler(ServerWorker)
onRegister(Attributes):boolean
onLogin(Attributes):void
startElement(String,String,String,Attributes):void
<<Java Class>>
ServerMain
server
ServerMain()
main(String[]):void
<<Java Package>>
pdus


<<Java Class>>
AckPDU
pdus
type: String
AckPDU()
AckPDU(String)
toXML():String
XMLParse(String):AckPDU
<<Java Class>>
UpdatePDU
pdus
UpdatePDU()
toXML():String
XMLParse(String):UpdatePDU
<<Java Class>>
RegisterPDU
pdus
username: String
password: String
RegisterPDU()
RegisterPDU(String,String)
toXML():String
XMLParse(String):RegisterPDU
<<Java Class>>
LogoutPDU
pdus
LogoutPDU()
toXML():String
XMLParse(String):LogoutPDU
<<Java Class>>
LoginPDU
pdus
username: String
password: String
LoginPDU()
LoginPDU(String,String)
toXML():String
XMLParse(String):LoginPDU
<<Java Class>>
FollowingUsersRequestPDU
pdus
FollowingUsersRequestPDU()
toXML():String
XMLParse(String):FollowingUsersRequestPDU
<<Java Class>>
UnfollowPDU
pdus
username: String
UnfollowPDU()
UnfollowPDU(String)
toXML():String
XMLParse(String):UnfollowPDU
<<Java Class>>
UserPDU
pdus
username: String
UserPDU()
UserPDU(String)
toXML():String
XMLParse(String):UserPDU
<<Java Class>>
ErrorPDU
pdus
reason: String
ErrorPDU()
ErrorPDU(String)
toXML():String
XMLParse(String):ErrorPDU
<<Java Class>>
PDU
pdus
PDU()
toXML(Class):String
XMLParse(String,Class):PDU
<<Java Class>>
ListEndPDU
pdus
ListEndPDU()
toXML():String
XMLParse(String):ListEndPDU
<<Java Class>>
PostPDU
pdus
text: String
author: String
date: String
likes: Integer
id: Integer
PostPDU()
PostPDU(String,String,Integer,String,Integer)
toXML():String
XMLParse(String):PostPDU
<<Java Class>>
UserListRequestPDU
pdus
UserListRequestPDU()
toXML():String
XMLParse(String):UserListRequestPDU
<<Java Class>>
LikePDU
pdus
postid: Integer
LikePDU()
LikePDU(Integer)
toXML():String
XMLParse(String):LikePDU
<<Java Class>>
UserContentRequestPDU
pdus
username: String
UserContentRequestPDU()
UserContentRequestPDU(String)
toXML():String
XMLParse(String):UserContentRequestPDU
<<Java Class>>
FollowPDU
pdus
username: String
FollowPDU()
FollowPDU(String)
toXML():String
XMLParse(String):FollowPDU
<<Java Class>>
PublishPDU
pdus
text: String
PublishPDU()
PublishPDU(String)
toXML():String
XMLParse(String):PublishPDU
<<Java Class>>
UnlikePDU
pdus
postid: Integer
UnlikePDU()
UnlikePDU(Integer)
toXML():String
XMLParse(String):UnlikePDU
<<Java Class>>
ListBeginPDU
pdus
type: String
ListBeginPDU()
ListBeginPDU(String)
toXML():String
XMLParse(String):ListBeginPDU