JAVA Software Workshop
Group Project Presentation

Athens

Benedict Churchus
Zsolt Pazmandy
Raymond Tjong
Hua Wei
George White

Messenger App

We've built an IM application for UoB SoCS users where students & staff can register, and chat in private rooms or groups. All chat history and user information is securely encrypted and stored in the CS database.

Outline

Structure:

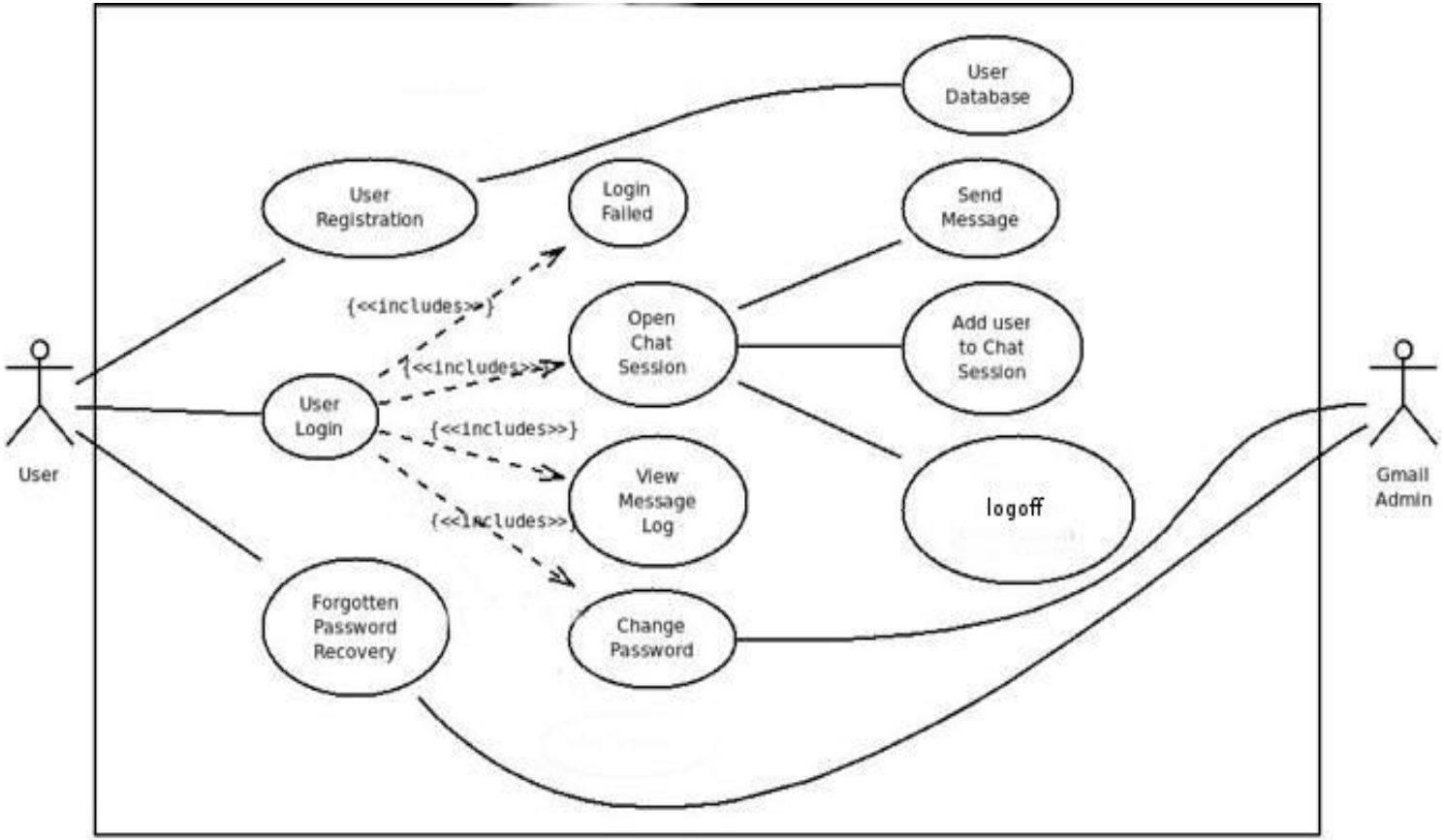
- All database communications done through the server
- No data stored persistently client-side
- Encrypted communications between server & client

Features:

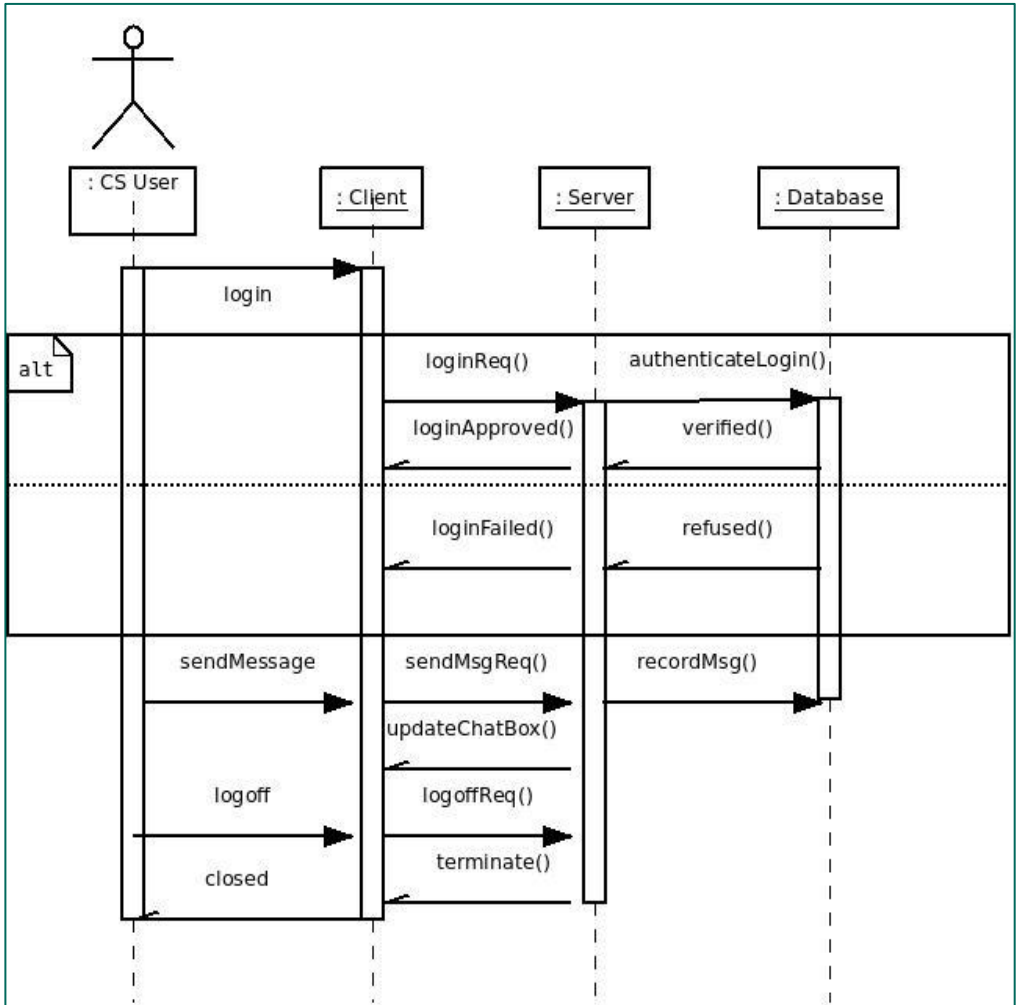
- User registration & editing existing profiles
- Chat logging & filtered viewing
- User status (available, busy, hidden)
- Forgotten password recovery through bham.ac.uk email

Design

Use Case Diagram



Design



Sequence Diagram:

A registered user logs in, sends a public message, and finally logs out.

Server

1. **Server initialized with:**
   - Server Socket
   - UserList (in Form of ArrayList<User>)
   - Thread pool (in order to avoid races)
2. **Listening in a port number**
   - Accept the socket
   - For login request, add this socket and user name to user list
3. **Communicate with clients**
   - Process requests
   - Response to one or more than one clients

Encryption

1) **Rail Fence Cipher (1st Encryption)**
   a) Plain text is put into a square matrix row by row
   b) Some random letter is appended in order to fill the square
   c) The matrix is read column by column
2) **Monoalphabetic Substitution Ciphers (2nd Encryption)**
   a) Normal table: [0-9]-->[0-9] [10-35] -->[A - Z] [36-61]-->[a - z]
   b) Substitution table: e.g. if the key is [Athens]:

```
'A' 't' 'h' 'e' 'n' 's' ⟶ 10 55 43 40 49 54
```

   c) Remaining slots in the matrix filled with salt
   d) Letters in plaintext substituted with numbers

Client

- Upon successful login, the client class is instantiated and all relevant user data is downloaded from the database
- A separate thread is started that listens for any response sent from the server for as long as the user is online
- The client stores the UserList (incl. all online users' relevant information), MessagesList (incl. sender, recipient & timestamp data) dynamically. All data is disposed of on logout.
- Data transmitted between Client & Server is converted into Strings which are always encrypted before transmission, decrypted on arrival and rebuilt on the receiving end.

Communications Protocol

REQUEST	PROTOCOL (Client to Server)	RESPONSE (Server to Client)
verifyRegistration	“2” + “req” + username	“2” + “res” + “0” / “1” / “2”
sendPrivateMessage	“3” + “req” + “0” + getUsername() + recipient + message	“3” + “res” + “1” + username + message
sendPublicMessage	“3” + “req” + “1” + getUSername() + message	“3” + “res” + “0” + message
changePw	“4” + “req” + getUsername() + getPw1() + getPw2()	“4” + “res” + “0” / “1” / “2”
viewPrivateLogs	“5” + “req” + “1” + getUsername() + getUsername2()	“5” + “res” + “1” + privateLog
viewPublicLogs	“5” + “req” + “0” + getUsername()	“5” + “res” + “0” + publicLog
changeStatus	“6” + “req” + getUsername() + status	“6” + “res” + username + “:” + status
forgottenPw	“7” + “req” + username	“7” + “res” + “1” / “2” + “0” / “1”
changeNickname	“n” + “req” + username + newNickname	“n” + “res” + username + nickname / “0”
login	“1” + “req” + username + password	“1” + “res” + “0” / “1”
logoff	“0” + “req” + username	-

Database

Database tables:

1. **‘Usernames’** table stores all valid CS usernames
   - `usernames` **`varChar(7) PRIMARY KEY`**

2. **‘Registered’** table stores all user profile data
   - `username` **`varChar(7) PRIMARY KEY`**
   - `nickname` **`varChar(20)`**
   - `pw_hash` **`varChar(40)`**

3. **‘Messages’** table stores all the messages
   - `msg_id` **`varChar(30) PRIMARY KEY`**
   - `message` **`varChar(4000)`**

4. **‘AccessTo’** table stores information on which user has access to which message
   - `username` **`varChar(7) REFERENCES`** `registered(username)`
   - `msg_id` **`varChar(30) REFERENCES`** `messages(msg_id)`

Register Model

- Standalone function, distinguished from client class
- Functionalities that even users who are not logged in can access
- Register method called in Register Controller
    - Establishes a socket to server
    - Encrypts request and send to server
    - Request includes username, nickname, password
    - Decrypt response from server
    - Returns an integer to indicate the outcome of process to Register Controller

Password Model

- Standalone function
- Implements change password functionality (accessed through settings after login) and
- Forget password
  - Both methods require a socket to communicate with server
  - Encrypts a request to server as per protocol and decrypts a response
- Forget password process has two layers
  - Checks username with database first
  - Check string input with locally stored string from response
  - Last stage allows user to change to a new password

Graphical User Interface (GUI)

- Implemented individual classes for each of the individual screens/windows
- Each controller class utilises methods from the model and client
- Update method makes sure all clients receive appropriate messages
- Listeners send user input to where it is required e.g. updating user’s nickname in the database

Testing

- Reliant on console output
- Tested through the path of a user
- Server side tested with JDBC
- Client Side working through the classes
- System.out.println()
- Manual GUI

Report

- Design rationale and scope
- Focus on system design
- Risk assessment and use cases
- Diagrams, UML and generic
- Team organisation and minutes
- Evaluation