Chatty

Introduction

Chatty is a point-2-point online chatting software.

Development Environment

Java:

```
java version "1.8.0_102"
Java(TM) SE Runtime Environment (build 1.8.0_102-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.102-b14, mixed mode)
```

IDE: Intellj IDEA

Framework Used

- Netty
- Hibernate
- Java Swing
- MySQL -- database

Function Implemented

- Sign up
- Sign in
- Sign out
- Add friend
- Send message to one specific person

Usage

Run as server

Before you start : Data persistence is based on MySQL, if you wish to run server on your own computer, make sure you have installed MySQL

Client does not need to set up database

- If you are using jar file, the root user password of your database should be set to `123456`
- If you are using source `.class` file, find file `com/ca/entities/hibernate.cfg.xml.tld`, edit

```
<property name="hibernate.connection.username">Your Own User</property>
<property name="hibernate.connection.password">Your Own Database Passwo
rd</property>
```

Also, create a database called `chat`, then use following SQL scripts to create tables

```
CREATE TABLE chat.User
(
    email VARCHAR(30) PRIMARY KEY NOT NULL,
    password VARCHAR(32) NOT NULL,
    registerTime VARCHAR(30),
    gender INT
);
CREATE UNIQUE INDEX User_email_uindex ON chat.User (email);

CREATE TABLE chat.Msg
(
    id INT PRIMARY KEY AUTO_INCREMENT,
    data VARCHAR(1000),
    source VARCHAR(30) NOT NULL,
    dest VARCHAR(30) NOT NULL,
    sendTime VARCHAR(30),
    CONSTRAINT fk0 FOREIGN KEY (source) REFERENCES User (email),
    CONSTRAINT fk1 FOREIGN KEY (dest) REFERENCES User (email)
);
CREATE UNIQUE INDEX Msg_id_uindex ON chat.Msg (id);

CREATE TABLE chat.Friends
(
    id INT PRIMARY KEY AUTO_INCREMENT,
    email1 VARCHAR(30) NOT NULL,
    email2 VARCHAR(30) NOT NULL,
    CONSTRAINT fk_0 FOREIGN KEY (email1) REFERENCES User (email),
    CONSTRAINT fk_1 FOREIGN KEY (email2) REFERENCES User (email)
);
```

Start Server Command:

- If you are using `.class` files, after setting up database correctly, open terminal, type

```
cd *root_of_your_.class_directory*
```

```
java Main Server
```

If everything is fine, you will see the last line of output as

```
Session Factory initialization finished
```

- If you are using `.jar` files, after setting up database correctly, open terminal, type

```
cd *directory_of_jar_file*
java -jar ca.jar Server
```

If everything is fine, you will see the last line of output as

```
Session Factory initialization finished
```

Start Client Command:

- If you are using `.class` files, open terminal, type

```
cd *root_of_your_.class_directory*
java Main Client
```

- If you are using `.jar` files, open terminal, type

```
cd *directory_of_jar_file*
java -jar ca.jar Client
```

To get connected to server, you need manually enter the IP address of existing server, make sure someone is running a server

Here is a server I am running (***May be changed every time I restart the remote AWS***) :

Source Code Compilation

If you wish to compile source on your own computer, please use Maven to download all dependencies

```
 <dependencies>
```

```
        <dependency>
            <groupId>io.netty</groupId>
            <artifactId>netty-all</artifactId>
            <version>4.1.34.Final</version>
        </dependency>

        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.4.1.Final</version>
        </dependency>

        <dependency>
            <groupId>org.json</groupId>
            <artifactId>json</artifactId>
            <version>20180813</version>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>6.0.6</version>
        </dependency>
    </dependencies>
```

After downloading all required external libraries, put all libraries in the same folder of the source files, then use command

```
cd *directory_of_source_files*
javac -Djava.ext.dirs=. *.java
```

If everything goes right, you will get all `.class` files you need, run them as mentioned in **instruction** part.

However I strongly recommend to use the jar file

Implementation Details

Overall

The software is developed using MVC thought, separate network module, database access layer and GUI module. There are two main parts, the client side and server side

The project is mainly focused on the logic behind the GUI, I am not a very good designer of GUI sadly

Interesting points:

- Object Relational Mapping
- Thread pool programming
- Protocol design
- Netty framework
- Design patterns (Singleton etc.)

Server

First comes the question: What should a functional server do so that it can provide qualified service to users?

There are two parts: stable network connection and functional data storage

For network connection, Java provided useful APIs and good native programming framework. Based on those, a lot of third party libraries appear. The famous structure Netty gives us convenient access to the network world.

Netty is an event-driven and non-blocking network programming framework.

For Netty tutorial, please see here:

```
https://netty.io/wiki/user-guide-for-4.x.html
```

As for storage, different databases provide different "dialect". However, hibernate solved this problem perfectly and provide transaction control. In addition, it makes it possible to save "entity" into database - object oriented programming :)

For Hibernate tutorial, please here:

```
https://www.tutorialspoint.com/hibernate/
```

In addition, creating new threads or destroying an existing thread is very expensive and cost a lot of system resources, so all multi-thread tasks are submitted to a fixed length thread pool

Clients

Function for clients should be process messages and send messages to others, give basic GUI to users.

Same as the server, Netty is used for network part.

Java Swing is used for GUI.

POJO:

POJO stands for plain-old-java-object, it does not have any special logic inside, but only the data that needed to represent an object.

In our case, we have two POJOs, Msg and User (not used)

Msg is designed to represent a complete message

User was initially designed to represent a user entity, however, it is replaced by UserEntity

ORM:

ORM stands for object-relational-mapping. Hibernate parsed the .xml file to find all mappings from POJO to database schema.

Three entities were implemented.

FriendsEntity is for representing a pair of friend.

MsgEntity is for representing a message sent by someone.

UserEntity is for representing a user.

Communication

Client and server should "speak the same language so that they could understand each other". In this sense, the "language" is what we say "protocol".

There are two kinds of message that a client will send to server:
1. Some message needed to be processed by server and give back result, like logging in
2. Some message needed to be redirected to another user and give back result

The message format for a complete package looks like following:

```
0           4              8           8+length
+------------+--------------+------------+
|Message type|Message Length|Message body|
+------------+--------------+------------+
```

After the server receives a message, certain job will be done based on Message type. Also, the message body is designed in JSON format in order to minimize the data amount transferred on network links.

Sign up:
Server parses user information, creates a UserEntity, saves into database

Login:
Server parses user email and password, checks data with values stored in database, sends back result (success or failure)

Get friend list:
This will only happen after login, server parses user email, wraps them into a JSON object, sends back to client

Add friend:
This will only happen after login, server parses user information, creates a FriendsEntity, saves it into database

Log out:
This will only happen after login, server will remove the client email and corresponding channel from a manager called *Online*

GUI:

A basic graphic user interface is provided.

The left is the place for sending and receiving messages, and the right is the place for user control.

Before sending message, you need to login first or sign up first (after creating a new account you will automatically login). Then your friend list will pop up at the bottom right corner.

Then choose one friend as your target user, type your message and send to him/her. However, it is not possible to send a message to someone that is not online.

If you want to add someone as your friend, simply login, add a friend using his/her email address, everything's done.

Future direction:

1. Be able to send message to someone offline
2. Be able to show online/offline status for friends
3. Better database design

4. Prettier GUI design
5. Group chat