CN Final Project (Team2)

2017.01.14

0. Environment

- Machine: Linux 64 bit / mac OSX 10.12
- Structure: Web server
    - Front-end: Jade + javascript
    - Back-end: Node.js + Express + npm
    - Database: Mongodb

1. Members & Team work

StudentID	Name	Work
B03902031	詹郁珍	File Transfer, Bonus x 1, Report
B03902037	陳郁棋	Messaging, Report
B03902024	鄭筱樺	Registration, Bonus x 5, Report

2. How to run

```
MUST BE EXECUTED IN /src

For linux:
./Build.sh

For mac:
./Build_mac.sh
```

3. Implement Details

3.1 Registration

1. Database
    a. Username
    b. Userpasswd
    c. customizeMsg
2. Signup: A User invokes HTTP POST to specify (username, password), and the server routes the request to /signup with a callback function that inserts (username, password) to database if the username doesn't exist, the customizeMsg is default to ""(empty string)
3. Login: A User invokes HTTP POST to specify (username, password), and the server routes the request to /login with a callback function that finds (username, password) in database to check if there is a matching

3.2 Messaging

1. Database
    a. Sender
    b. Receiver
    c. Important flag: Users can label this flag if they thinks the message is important.
    d. Read flag: Records whether the message has been read by receiver.
    e. Message Content
2. Select a receiver and show history and : A user invokes HTTP POST to specify which user it wants to contact, and the server routes the request to /msgSend with a callback function that fetch all messages between currentUser and receiver in database. Also, set all unread messages that received from the receiver to read.
3. Send message: A user invokes HTTP POST to specify the message content and the important flag, and the server routes the request to /msgSendContent with a callback function that inserts the information into database with read flag = 0
4. Since we store message in database, messages will persist after the server restarts and there is no online/offline message sending problem
5. We use "url query" to allow more than one user to send message simultaneously
6. Refresh message sending page every 10 seconds for users to see the new messages.

3.3 File Transfer

1. Database
    a. Sender
    b. Receiver
    c. Filename
2. Send File: A user invokes HTTP POST to specify which file to transfer, using html type file input. All the type file input will send packed in req.file, and the server could get the file by req.file.[filefield_name].(ex. If the file_field's name is "uploadFile", then the server can get the file by req.file.uploadFile).
3. By using a api named "express-fileupload", we can mv the file to the destination directory on server and rename it [sender]_[receiver]_[filename], the store it in database.

4. Get what files I received: the action is bond with Messaging → we get the filecollection in db, and choose those sender matches currentUser and receiver matches receiver, then at the display page, we cut the filename(which is [sender]_[receiver]_[filename]) to only [filename] and display it in the file div.
5. Download file: click the file you want to download then it will download. Implement by sendFile function which is included in express. We tell the server where the file is then send the file to user. If it is something that can display on web page, then it will display and the user can decide whether he/she wants to download it; else, it will download automatically.

4. User and Operator guide

Step 1. User can sign up/login/change password, shown in Figure 1. We also set two button "showDB"and "cleanDB" to check the data which we store in the database and clean all the data in the database, shown in Figure 2.

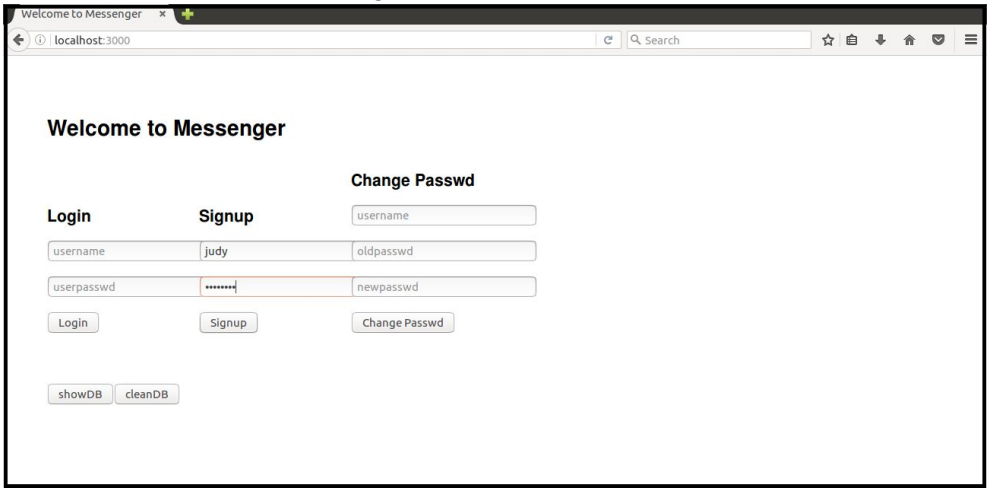

Figure 1: Start interface

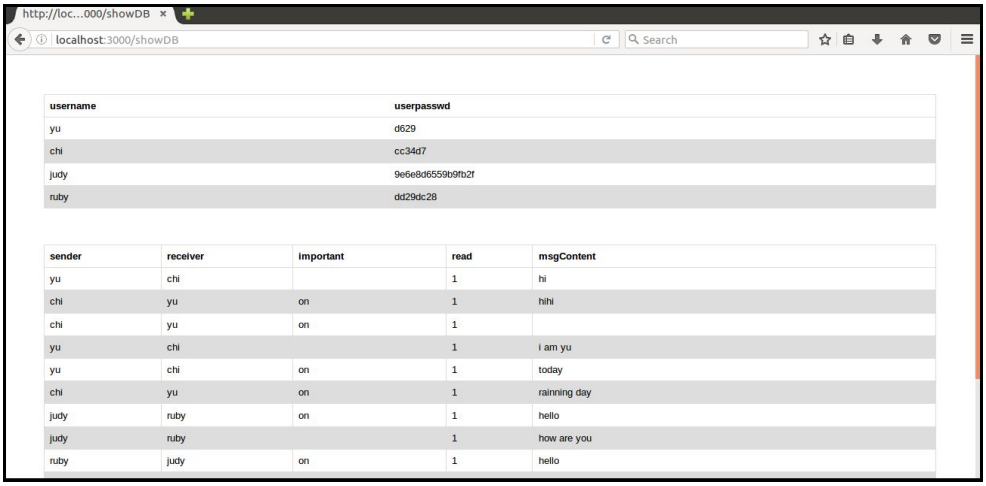

Figure 2: Show the content of our database.
It is notable that our userpasswd is shown after encryption.

Step 2. After the user login, he/she could see his/her unread messages and some importantDoc

marked by other user, shown in Figure 3. Besides, he/she could enter receiver's name to see the historical messages and send messages to the receiver. We also add the function that could search the certain message content, shown in Figure 4.

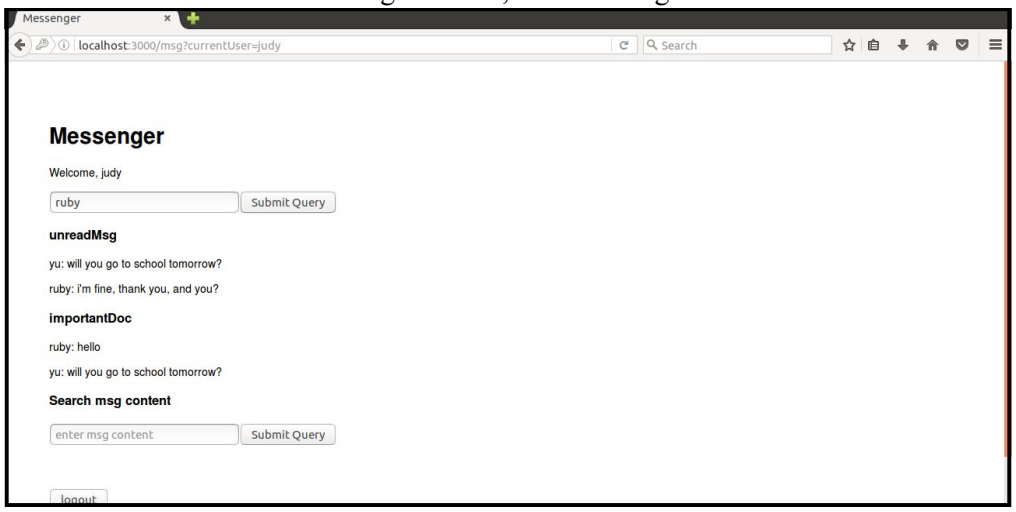

Figure 3: User can enter receiver's name as well as see unreadMsg and importantDoc.

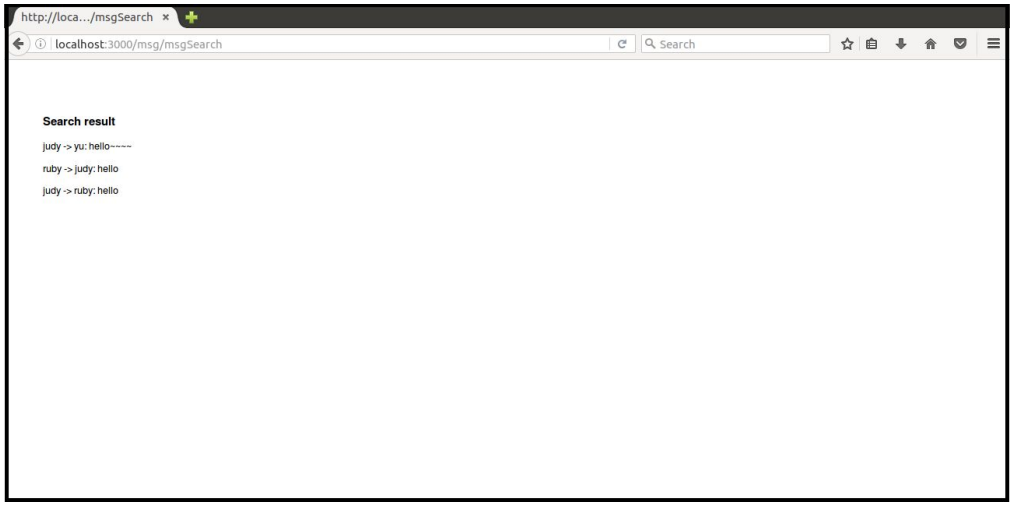

Figure 4: Example for searching the word "hello"

Step 3. After entering the receiver's name, he/she could start to send messages or files, shown in Figure 5. We also add a function that could add own message, shown in Figure 6.

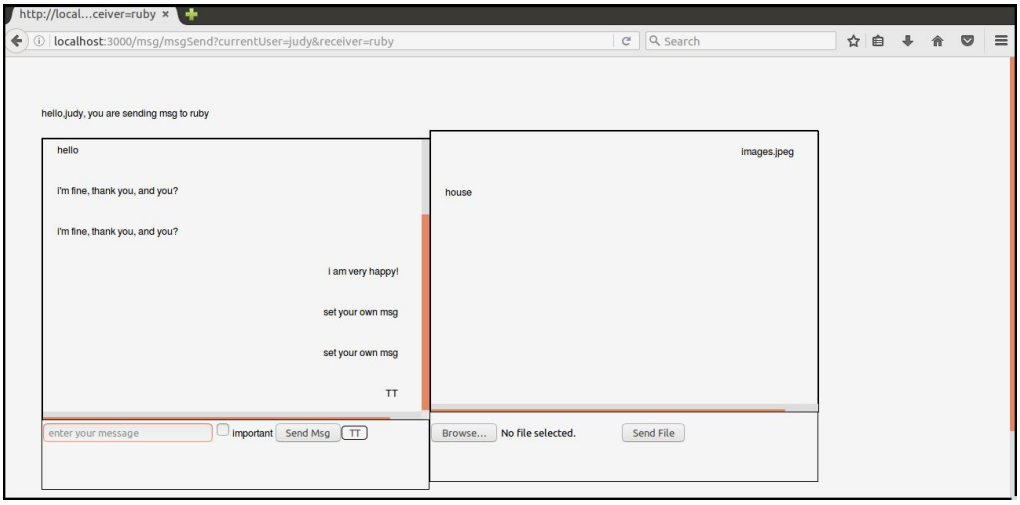

Figure 5: The left is the message part and the right is the file transmission part. On the message part, user can send messages and decide whether it is important or not. On the file transmission part, user can choose the file to send. Also he/she can download the file by clicking the filename.

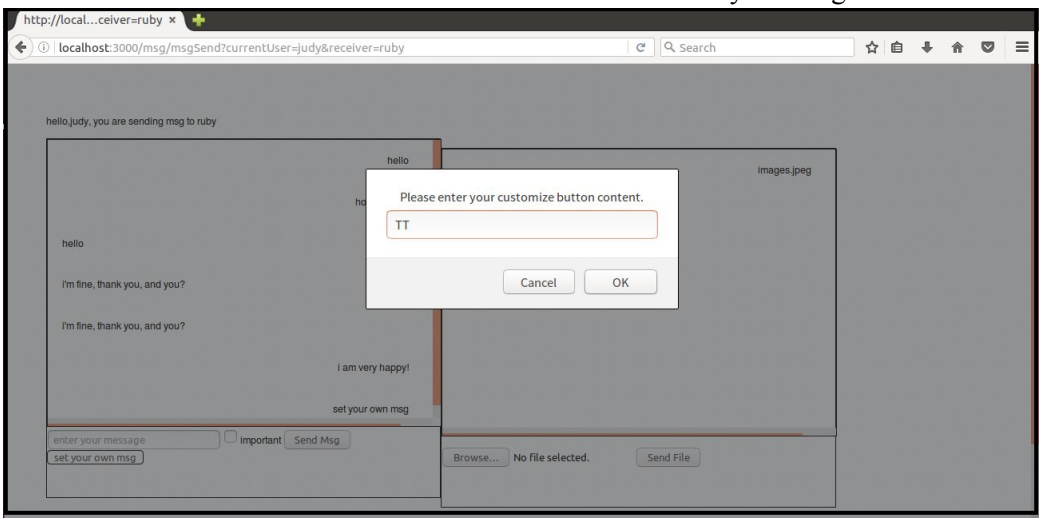

Figure 6: Right click “set your own msg” button and you can set customized button content which helps you send the msg you set just by a left click, without typing any words.

5. Bonus

5.1 Password encryption

1. Use npm package “crypto” to encrypt password before storing into the database
2. Compare input password in the database after encrypt it when users login

5.2 Change password

1. A User invokes HTTP POST to specify (username, old password, new old password), and the server routes the request to \changpasswd with a callback function that finds (username, old password) in database to check if there is a matching, and update password with a new one.

5.3 Show unread messages

1. Read flag is set to 1 when a user visit to \msgSend page
2. Find all messages whose receiver equals to the current user and have a read flag 0.
3. Show them.

5.4 Enable users to label important messages and show receiving important messages

1. Important flag is set to 1 if a user specifies the message to be important before sending a message.
2. Find all messages whose receiver equals to the current user and have an important flag 1.
3. Show them.

5.5 Search messages by keywords and show search results

1. A User invokes HTTP POST to specify the keywords
2. Find all messages whose sender is the current user or whose receiver is the current user and contains the keyword
3. Show all search results

5.6 Enable users to set a emoticon and send it by click

1. Add a customized message: First, we add a new button on page, and control it with jquery. If it is right-clicked, then we call the “prompt” function , then the user could set what message he/she wants to set. After enter the string and confirm it, it will send a HTTP POST request to server, named /msg/msgSend/customizeSet, and then it adds the customizeMsg to usercollection database. The customize message follows the currentUser, instead of receiver.
2. Send a customized message: just click the button ! After left-clicked, it will send a HTTP POST request to server, named /msg/msgSend/customizeSend, and then it adds the text of the button to the msgcollection in database. At last it will refresh the page.

6. Problems & Solutions

6.1 Asynchronous

Javascript executes code asynchronously, it will cause unexpected result if we assume it to run synchronously. Our solutions:

1. Use callback functions
2. Use forEach
3. Use an npm library called “async”

6.2 Multi-user

We don’t know how to implement multi-user interaction at first, because the login state will be replaced by the later login user. Our solutions:

1. Add query to every actions.
2. At every refresh, we get the currentUser and receiver by the query and this avoids the replace problem.