Computer Network Report

Team Members and Work Divisions

- B08902029 陳咏誼: client and browser
    - `client.cpp, index2.html, index2.css, main2.js`
- B08902071 塗季芸: server, console and database
    - `server.cpp, console.cpp`

Accomplishment

We satisfy **all requirements** with the following **Bonus**:

- Browser mode:
    - Upload text file through browser.
    - Click image to download image.

README (User Instruction)

Demo Link: https://youtu.be/QdFPxIA9WrI
Github Link: https://github.com/jiyuntu/NTU-Computer-Network-Chatroom

Compilation

```
$ make
$ ./server [port]
$ ./client [ip:port] [port2] // For browser mode
$ ./console [ip:port] // For console mode
```

Neccessary Files and Directories

Server

```
|   Makefile
|   server.cpp
|   sqlite3.c
|   sqlite3.h
```

```
|   sqlite3.o
|   sqlite
|   shell.c
|
+---server_dir
    |
    +---default/
      |   index2.html
      |   main2.js
      |   index2.css
      |   report.pdf
```

Client (Browser mode)

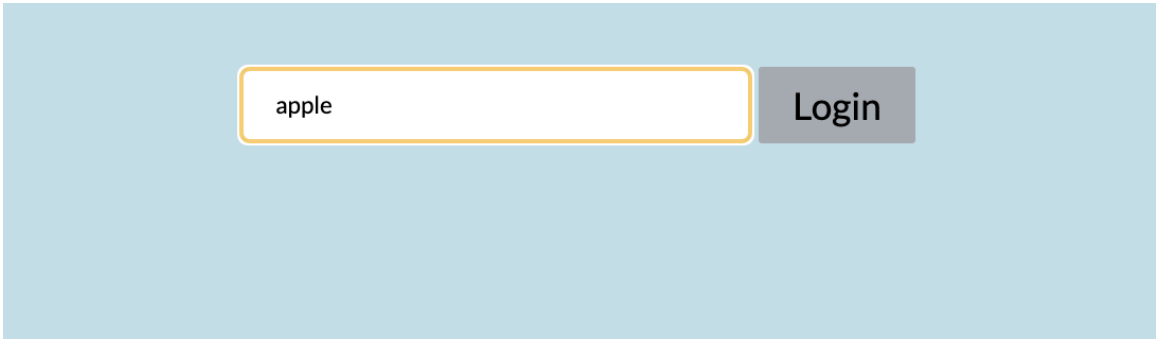
```
|   Makefile
|   client.cpp
```

Client (Console mode)

```
|   Makefile
|   console.cpp
|
+---client_dir
    |   any_file_to_be_put
```

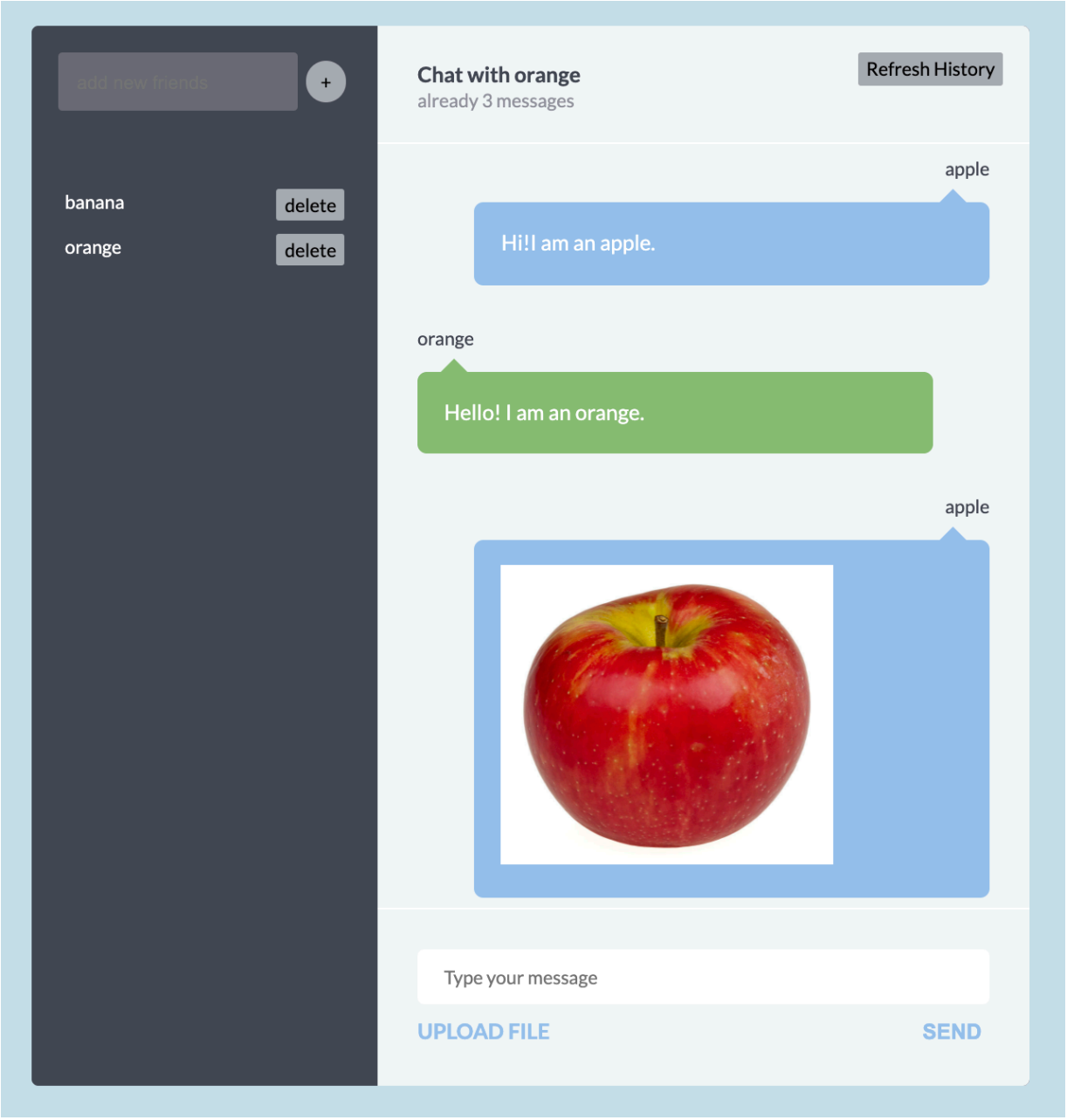
Browser Mode

1. Access `http://localhost:port2/`
2. Enter `$username` to login



3. Start chatting! You can
    - Upload text files (Bonus)
    - Click image or file link to download file
    - Add/Delete friend

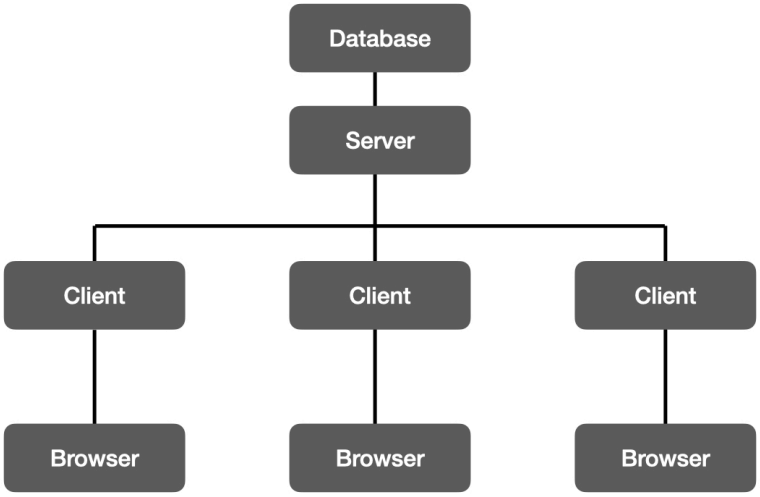
- Refresh history
  - Every time user sends a message the history would be refreshed.
- Send message
- Every input section allow uses press `enter` to send.



Console Mode

Follow the instructions in Communication between server and client
(Only need to type the instruction in the first line.)

Implementation



Note:

- Our design accepts single-way friendship.
  And we only shows the history after you add that person to your friend list.
- `client.cpp`: Browser mode.
  `console.cpp`: Terminal console mode.

Example of single-way friendship

Step1

Cathy's friend list: ["Wendy"]
Wendy's friend list: ["Bob"]

Step2

Cathy says "Hi, I'm Cathy" to Wendy.

Step3

Wendy adds Cathy as her friend.
Wendy's friend list: ["Bob", "Cathy"]

Step4

Wendy says "Hi, I'm Wendy" to Cathy.

Step5

On Cathy's screen:

```
Cathy -> Wendy: "Hi, I'm Cathy"
Wendy -> Cathy: "Hi, I'm Wendy"
```

On Wendy's screen:

```
Wendy -> Cathy: "Hi, I'm Wendy"
```

Communication between server and client

1. Add a new friend

```
client -> server: add $username $friendname
(Do not need a server response)
```

We assume that every possible name exists, so server doesn't need to response success or error.

2. Delete a friend

```
client -> server: delete $username $friendname
(Do not need a server response)
```

If `$friendname` is not in `$username`'s friend list, server just does nothing.

3. List all friends

```
client -> server: ls $username
server -> client: $response_length
client -> sevver: 1 (ACK)
server -> client: ["friend1", "friend2", ...]
```

4. Say something to a specific friend

```
client -> server: say $username $friendname $something
(Do not need a server response)
```

We don't accept `$something` including `\n`

5. Show the chat history with a specify friend

```
client -> server: history $username friendname
server -> client: $response_length
client -> sevver: 1 (ACK)
server -> client:
[{
"From": "$username",
"To": "$friendname",
"Content": "A"
},
{
"From": "$friendname",
"To": "$username",
"Content": {"File": "a.jpg"}  // The friend uploads a file
}]
```

6. Get a file

```
client -> server: get $username $filename
server -> client: $response_length
client -> sevver: 1 (ACK)
server -> client: #file_content
```

7. Put a file to a friend

```
client -> server: put $username $friendname $filename
server -> client: 1 (ACK)
client -> server: $file_length
server -> client: 1 (ACK)
client -> server: $file_content
```

Communication between server and database

Database Schema

Username	Friend	Content
VARCHAR(20)	VARCHAR(20)	Text

1. `add $username $friendname`
   add (username, friend, "") row
2. `delete $username $friendname`
   remove (username, friend) row
3. `ls $username`
   select all friends of username as

```
["friend1", "friend2", ...]
```

4. `say $username $friendname $something`
   update (username, friend, ?) to (username, friend, ? + something) using SQL UPDATE statement
5. `history $username friendname`
   print all chat history between username and friend as

```
[
{
"From": "username",
"To": "friend",
"Content": "A"
},
{
"From": "friend",
"To": "username",
"Content": "B"
},
{
"From": "Lisa",
"To": "Peter",
"Content": {"File": "a.jpg"}  // Lisa upload a file.
}
]
```

for commands like `ls` and `history`, we collect all response from database by

```
sqlite3_prepare_v2()
sqlite3_bind_int()
sqlite3_step()
sqlite3_column_text
```

Communication between client and browser

Steps

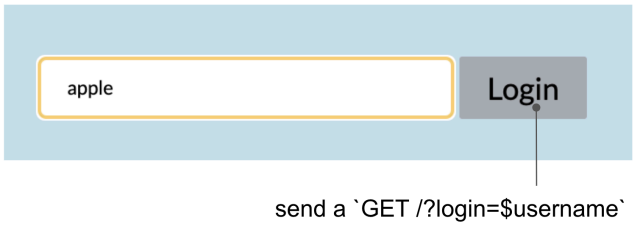
The process of satisfying a request is as following:

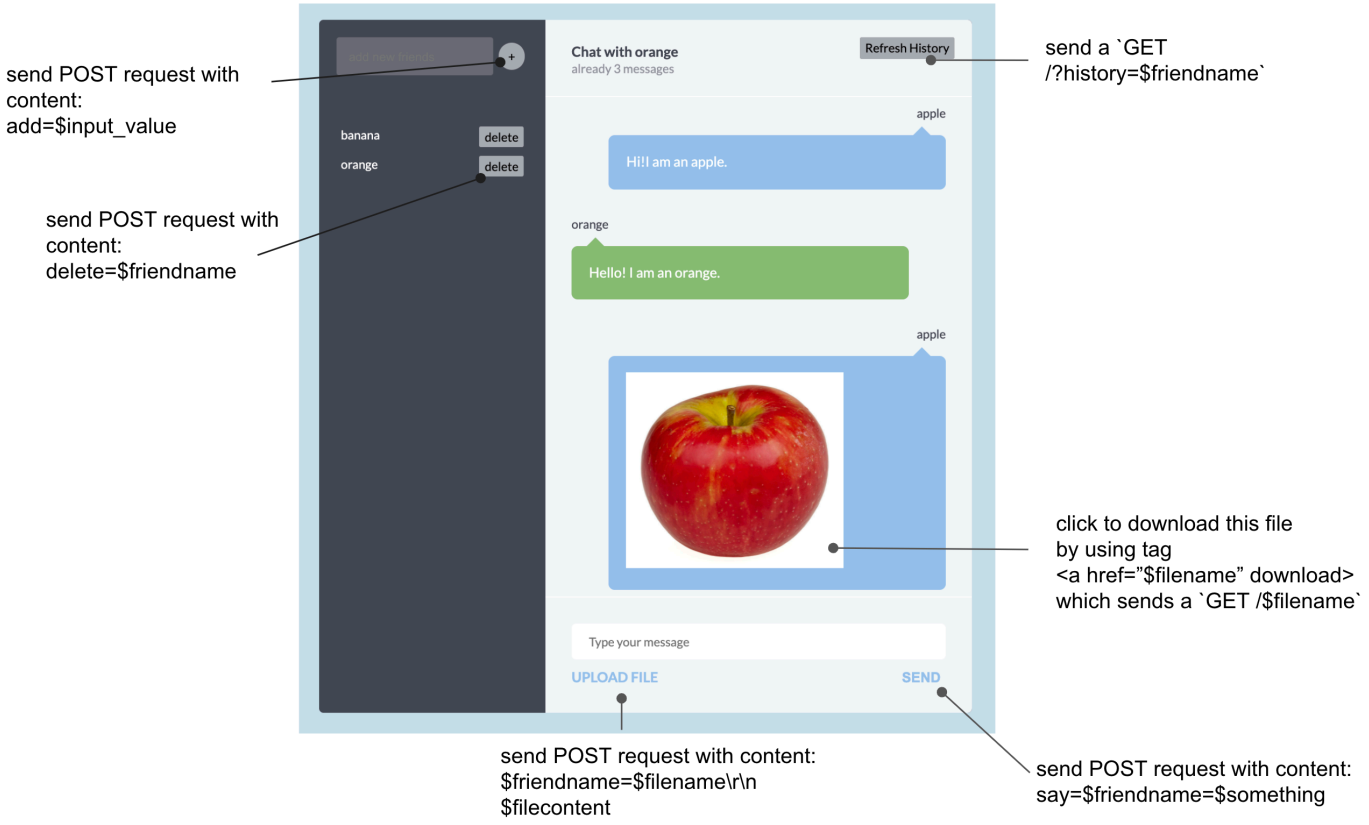
1. browser builds a connection to client.cpp.
2. client.cpp accepts the request from browser.
3. client.cpp transform the HTTP request to the format for communicating with server.
4. client.cpp receives response from server.
5. client.cpp transform the response to HTTP response and send back to browser.
6. client.cpp close the fd of the request.

Note that when the browser accesses `http://localhost:port2/` , it sends a `GET /` to client.cpp. The client.cpp sees the request as `GET /index2.html`.

We attach a `main2.js` and `index2.css` in `index2.html`.
The control logic of browser is written in `main2.js`.

Request transformation





GET

```
browser -> client: GET /?login=$username
client -> server: ls $username
```

```
browser -> client: GET /$filename
client -> server: get $filename
```

```
browser -> client: GET /?history=$friendname
client -> server: history $username $friendname
```

PUT(POST)

```
browser -> client: <POST
HEADER>\r\n$friendname=$filename\r\n$filecontent
cleint -> server: put $username $friendname $filename
```

OTHER POST

```
browser -> client: <POST HEADER>\r\n$say=$friendname=$something
cleint -> server: say $username $friendname $something
```

```
browser -> client: <POST HEADER>\r\n$add=$friendname
cleint -> server: add $username $friendname
```

```
browser -> client: <POST HEADER>\r\n$delete=$friendname
cleint -> server: delete $username $friendname
```

Server

For each client connection, assign a thread for it. Use a while loop to accept its commands, then query the database, and send back the results.