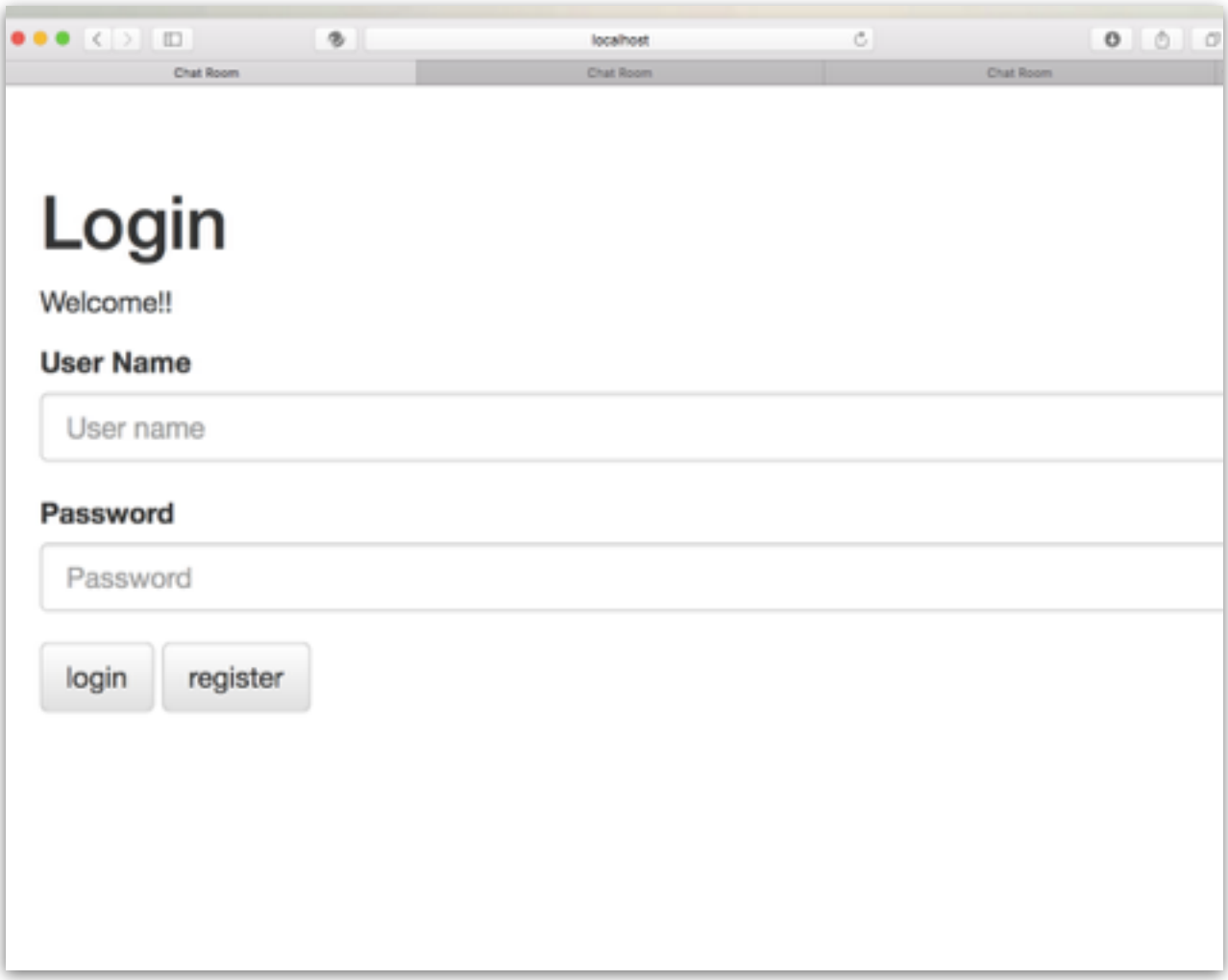


Project Report

Group 12
B02608032 Tsung-Hung, Hsieh
B03901191 Fu-Hsuan, Liu
Prof. Zhen-mou, Zheng

OUTLINE/FEATURES

Introduction

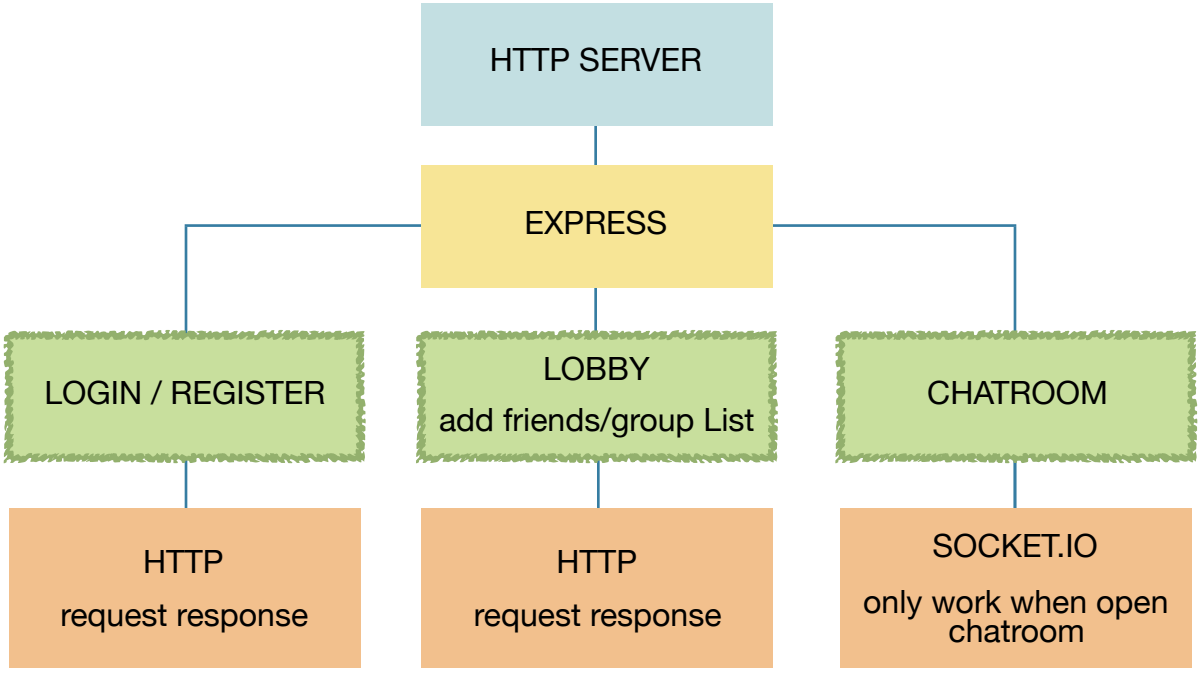
Our chatroom is based on socket.io and Express -node.js Web Application/API with these features below:

- Register and Login Interface
- Support 1v1 and group chat
- add friends freely
- Chat with yourself : provide cloud / storage service
- Can load the message before becoming friends

DESIGN CONCEPT

Server

Flow chart



Client

- BootStrap: CDN
- HTML: Hogan js template engine (.hjs)
- jQuery

Data Storage

We store data and information in *object* style —> .json at server.

app.register = {}; // {userName: password}
app.messageList = {}; // {userName: {groups: [messages]}

Valid Actions

app.loginAction = loginAction(app);
app.lobbyAction = lobbyAction(app);
app.chatroomAction = chatroomAction(app);

CODE STRUCTURE

Messenger
bin
node_modules
public
routes
util
views
app.js
package.json
run.sh

bin - www

www is the entry point:

- Declare the variations, module dependencies:

```
var app = require('../app');
var debug = require('debug')('messenger:server');
var http = require('http');
var socketio = require('socket.io');
```

- Get port from environment and store in Express.

```
var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);
```

- Create HTTP server using express app and setup socket.io

```
var server = http.createServer(app);
var io = socketio(server);
app.setSocketio(io);
```

- Listen on provided port, provide error handling functions.

```
server.listen(port);
server.on('error', onError);
server.on('listening', onListening);
```

- Normalize a port into number, string or false
- Event listener for HTTP server “Error”/“Listening”.

app.js

Arrange routing and deal with request when a client connect to server. Various actions and functions applied depending on the triggered events.

- Login Page: register, login
- Lobby: add friend, create group
- Chatroom: 1 to 1 chat, group chat

- require express module as the main application
- require several routers handling requests

- require action handling modules
- set parsers
- set view template engine
- setup socket.io

public - javascript

- Scripts that are executed at client side. Using jQuery.
- Data are sending by POST.
- Set event listeners to handle client actions server responses.
- Dynamically arrange the layout, the message color and alignment depend on if the client is sender or receiver.
- Set socket.io event listeners to send and receive messages.

public - stylesheets

- Created but turned out not used.
- Using bootstrap instead.

routes - chatroom.js / lobby.js / login.js

Provide routing functions that deal with http requests and responses that are sent to specific urls. These routers are implemented separately and thus a little bit easier to maintain.

Only Chatroom use socket.io, socket.io connection is only *turned on* when a client is routed to chatroom page. The server will assign a room to the user according to user's name and the friend or group chosen. All messages are only available to the users who are assigned to the same room.

util - chatroomAction.js / lobbyActions.js / loginAction.js

Define functions on server side to handle different events.

LoginAction:

checkRegister:

'name used'

'register successful'

checkLogin:

'not register'

'wrong password'

'login successful'

LobbyAction:

addGroup:

'added already'

'name not found'

'add successful'

getGroupList:

return groupList that the user is in

chatGroup:

redirect to the chatroom when user clicks on the group

ChatroomAction:

storeMessage:

store the new message in app.messageList

getOldMessage:

retrieve previous message

Implementation detail

Define KEY of chat information:

chat between two users: ‘FRIENDNAME’ (local to user)

chat groups use its id: ‘GROUPNAME—USER1-USER2-…-USERN ’ (sorted, global)

Actually, the 1 to 1 chatting is treated as group as well, with some modifications. It uses the friend’s name as the group name, and does not specify who are in the room (since only user and user’s friend).

The multi user chat room is identified by the group name and all the users’ name who are in the group, thus duplicate group name is allowed, and different group name with same users is allowed as well. Since user name is unique, we sort the users’ name in the group as identifier, and combine it with group name to get the unique group id.

views - .hjs and bootstrap

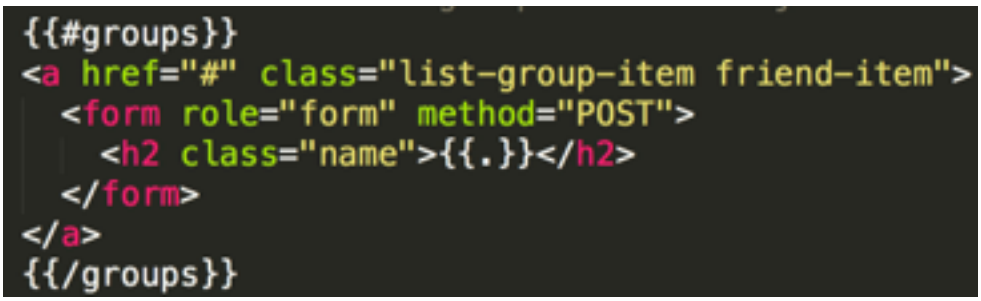
Hogan.js is a compiler for the Mustache templating language.

Using bootstrap by CDN.

In app.js :

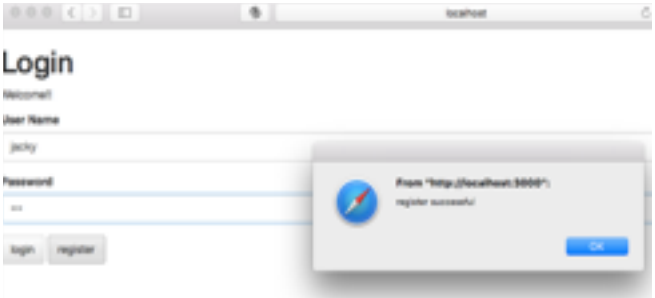
```
app.set('view engine', 'hjs');
```

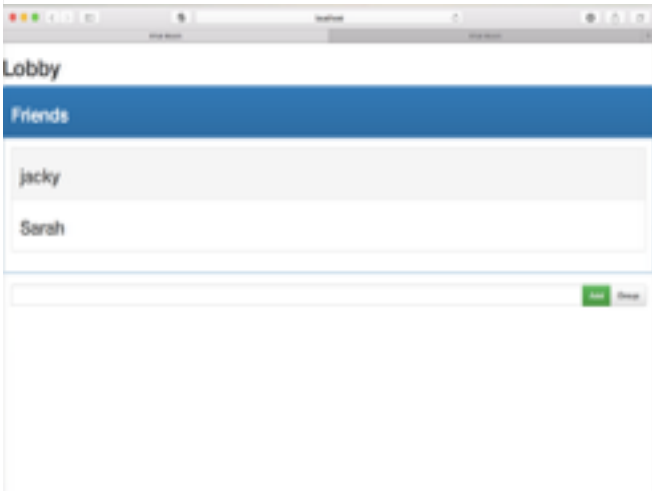
Make views/index.hjs :

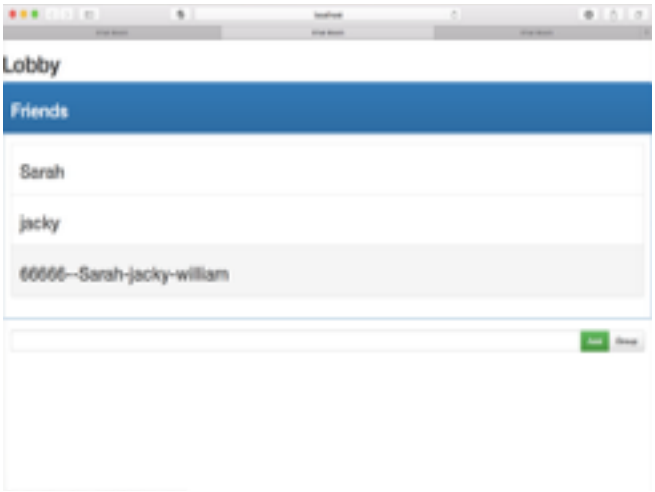


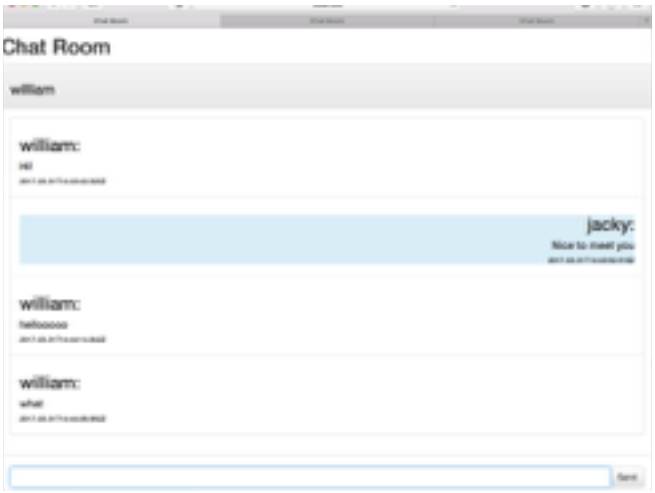
DEMO

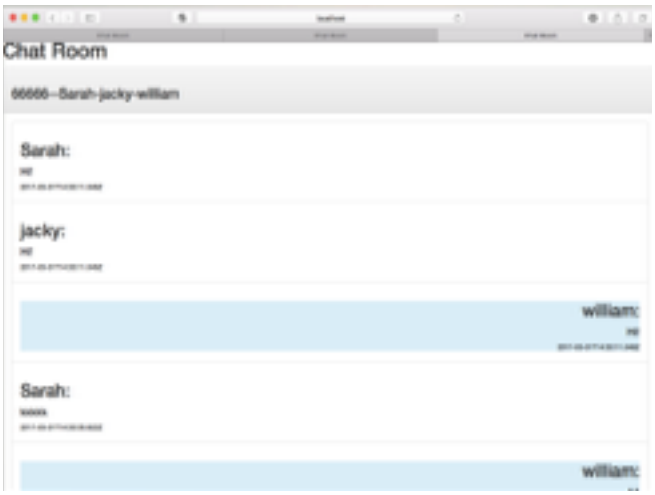
User interface











Terminal

{userName:”, password:”, action:”}
{friendName:”, action:”}

OLD MESSAGE: [{from:”,
to:”,
content:”,
timestamp:”}]

receive message: { roomName: ‘’,
from:”,
to:”,
content:” }

```
GET /lobby/javascripts/lobbyClient.js 304 1.288 ms - -
{ friendName: 'sarah', action: 'addFriend' }
POST /lobby/jacky 200 9.018 ms - 49
GET /lobby/jacky 200 4.784 ms - 3985
GET /lobby/javascripts/lobbyClient.js 304 3.857 ms - -
{ friendName: 'sarah', action: 'chatFriend' }
POST /lobby/jacky 302 11.462 ms - 86
GET /chatroom/jacky-sarah 200 3.674 ms - 2160
GET /chatroom/javascripts/chatroomClient.js 304 1.025 ms - -
a user connected~~
connect to:  jacky-sarah
FROM jacky TO sarah
HASH: jacky-sarah
OLD MESSAGE: [ { from: 'jacky',
    to: 'sarah',
    content: 'Hi~',
    timestamp: 2017-03-31T04:10:44.148Z },
  { from: 'sarah',
    to: 'jacky',
    content: 'Hi~',
    timestamp: 2017-03-31T04:10:44.148Z } ]
FROM jacky TO sarah
HASH: jacky-sarah
receive message:  { roomName: 'jacky-sarah',
  from: 'jacky',
  to: 'sarah',
  content: 'nice to meet you' }
{ friendName: 'jacky', action: 'addFriend' }
POST /lobby/sarah 200 10.056 ms - 49
GET /lobby/sarah 200 11.529 ms - 3985
GET /lobby/javascripts/lobbyClient.js 304 1.078 ms - -
{ friendName: 'jacky', action: 'chatFriend' }
POST /lobby/sarah 302 8.114 ms - 86
```

default welcome message

new message information

CONCLUSION AND DISCUSSION

Since we have no background in any web programming skills, things are not done elegantly, there are still many things can be improved.

Future prospects

- Data base: data will not disappear even server offline
- More complete, user friendly and beautiful user interface
- User can choose ‘Accept’ or ‘Decline’ when getting friend inviting notification
- Extra features: upload and download photos, videos, little games…etc

REFERENCE

[Bootstrap] http://getbootstrap.com/
[Express js] http://expressjs.com/en/4x/api.html
[Node js] https://nodejs.org/en/
[jQuery] http://api.jquery.com/
[Socket.io] https://socket.io/docs/
[Hogan js] http://twitter.github.io/hogan.js/
[JavaScript] https://developer.mozilla.org/en-US/docs/Web/JavaScript
[HTML] https://developer.mozilla.org/en-US/docs/Web/HTML
[CSS] https://developer.mozilla.org/en-US/docs/Web/CSS

[React js] https://facebook.github.io/react/docs/hello-world.html
[Babel js] https://babeljs.io/
[webpack] https://webpack.js.org/
[JSX] https://jsx.github.io/doc/tutorial.html
[Github Pages] https://pages.github.com/