INF
Informatik

Hochschule Reutlingen
Reutlingen University

Media and Communication Informatics

Cloud Computing WS 2017

Prof. Dr. Marcus Schöller

Submitted on

2018/01/12

Authors

Alexandros Konstantakos, 741590

Zahid Ibnu Yusuf, 741463

1 Requirements

These requirements were defined in the exercises. We split them into functional and non-functional ones, and defined a couple of our own marked in blue, of course with a lower priority.

1.1 Functional

1. A new user must register with the website first by providing a chat name.
2. A marked message is generated and sent to the chat room each time a new user connects to the chat room or leaves the chat room.
3. A user can request a list of all online users using the "\list" command; that list is only sent to the requestor.
4. A user can send a message to one dedicated recipient (private message) instead of to all online users in the chat room.
5. A user can send multi-media files to the participants of a chat room or via a private message, e.g., pictures, movies, sound files.
6. Implement the IBM Tone Analyzer in the chat.
7. Add an upload button to the registration dialog to let users provide a profile picture. Use the Bluemix "Visual Recognition" service to ensure that the picture contains a human face.
8. A user can press on a button to get information about the available commands.

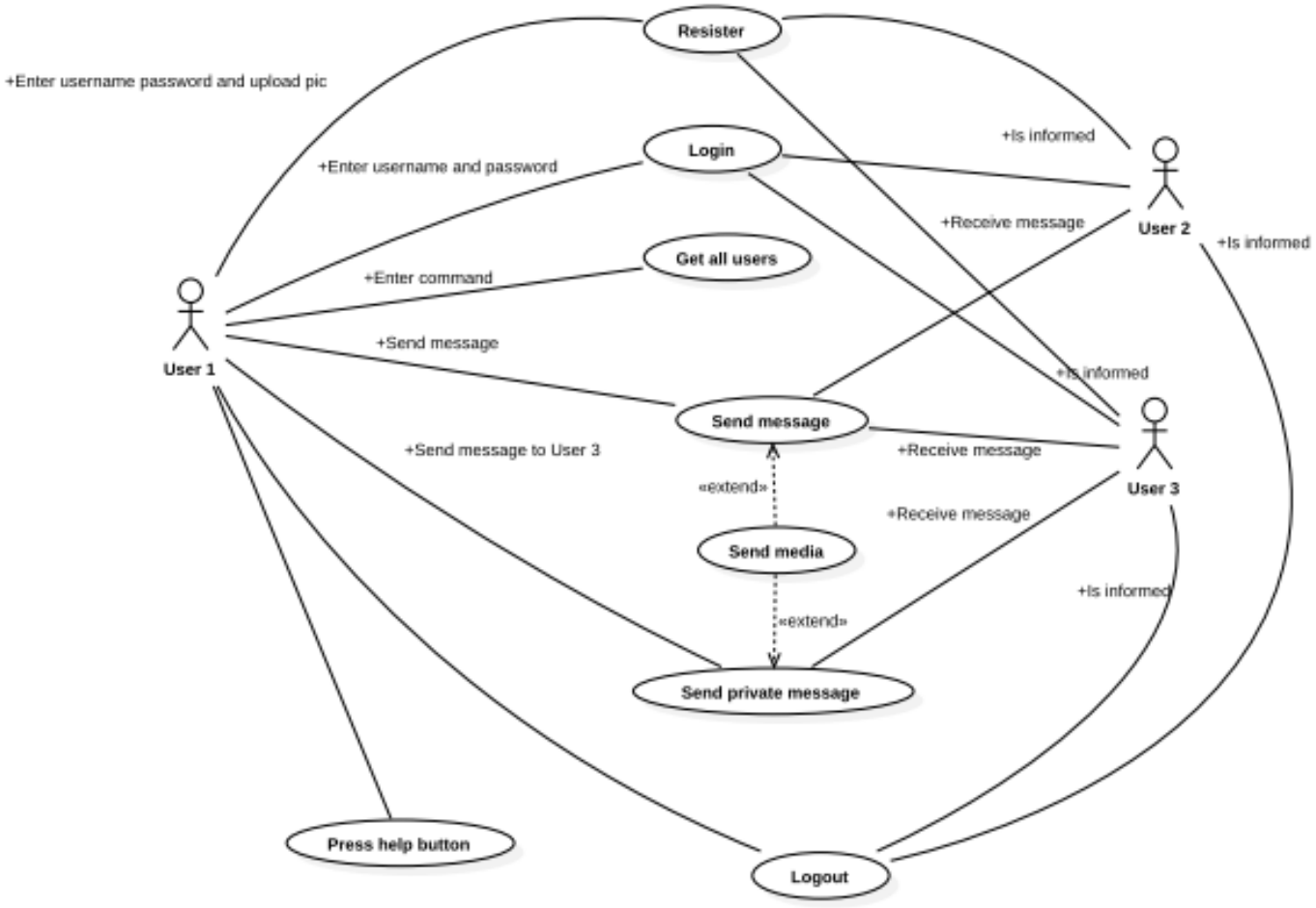
1.2 Non-Functional

1. In the chat room, all messages should be displayed in chronological order with a timestamp and the chat name of the user who has posted the message.
2. Enhance your registration dialog with a password field. The user credentials should be stored persistently and securely in a Bluemix data base. During the login process, the provided credentials should be validated against the data stored in the database.
3. Ensure data confidentiality and integrity by using TLS channels between client and server only.
4. Having deployed and tested your enhanced chat server, run a dynamic "Application Security on Cloud" test on your application and put the results in your presentation.
5. Fix at least three identified issues from requirement 4.
6. It should be a single page web application, to avoid reloading the page.

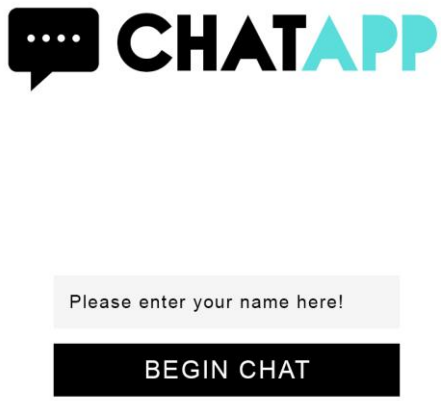
2 Design

We started our design of the application by creating a use case diagram and some mockups based on the requirements. After that we proceeded with the creation of a component diagram. Finally, to minimize merge conflicts and get a good workflow, we created a class diagram and splitted the source code into multiple files.

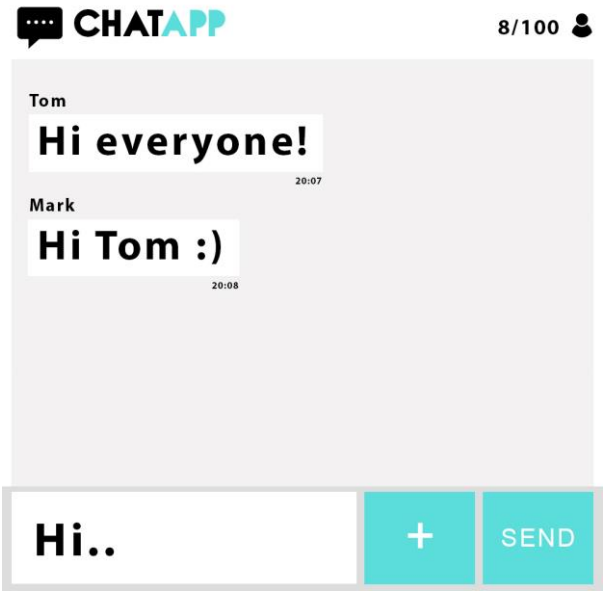
2.1 Use Case Diagram



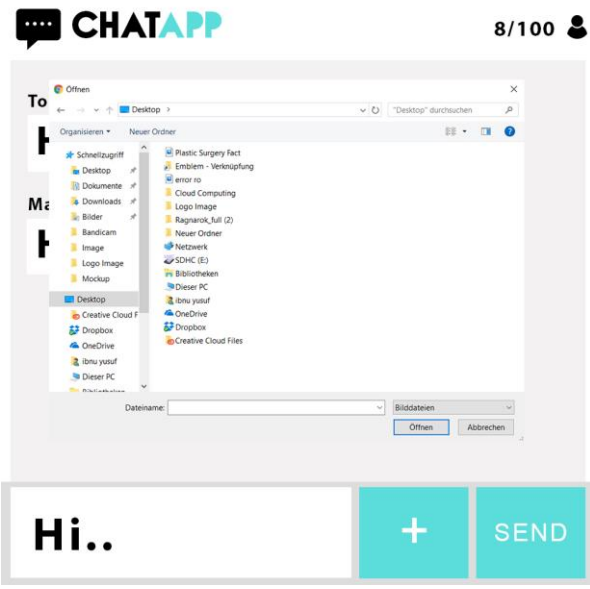
2.2 Mockups



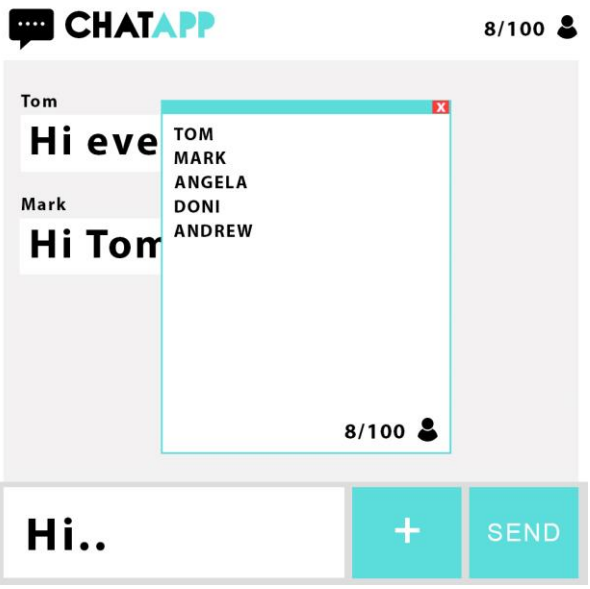
The Login Screen



The Chat Room

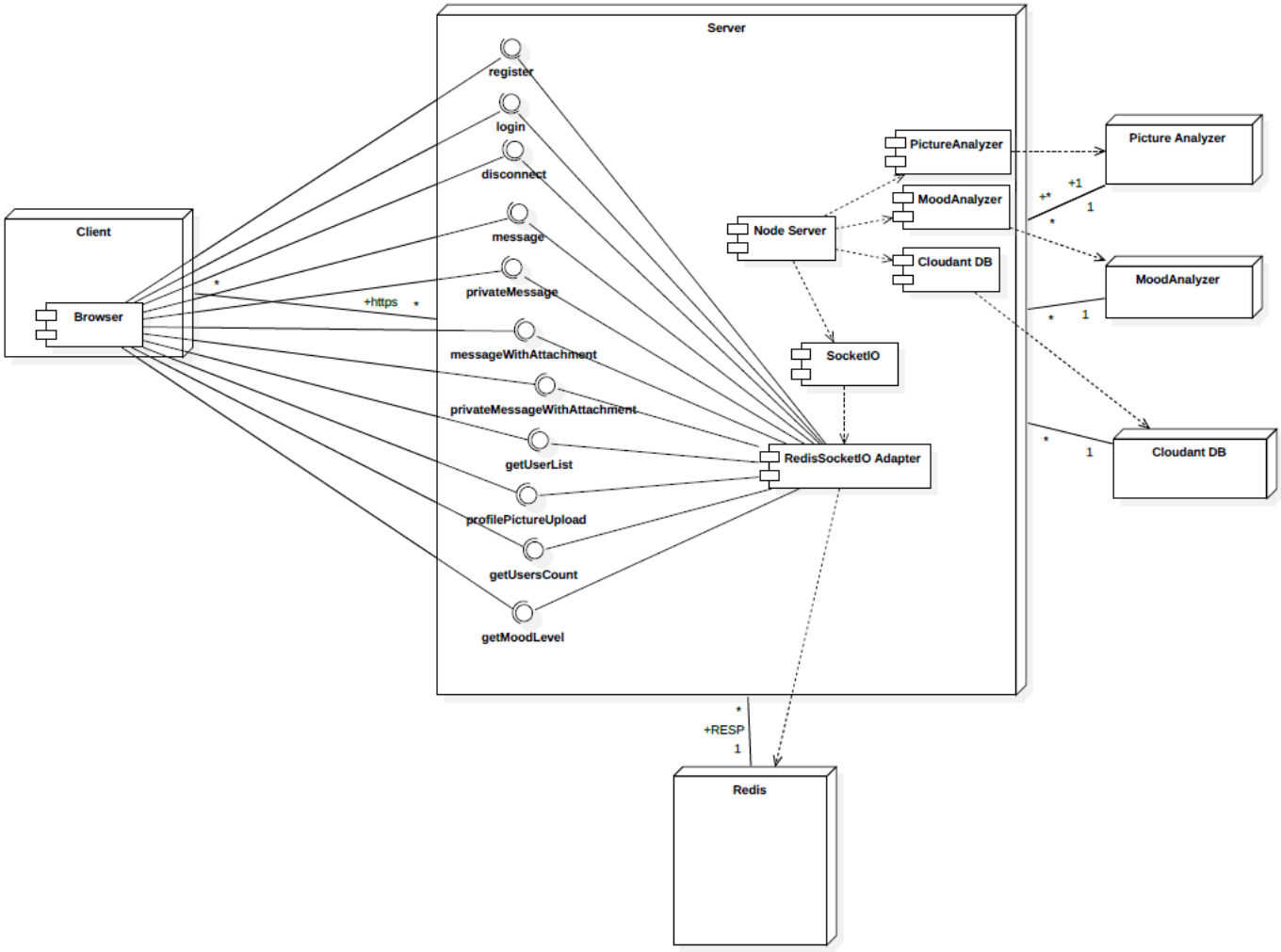


Uploading media

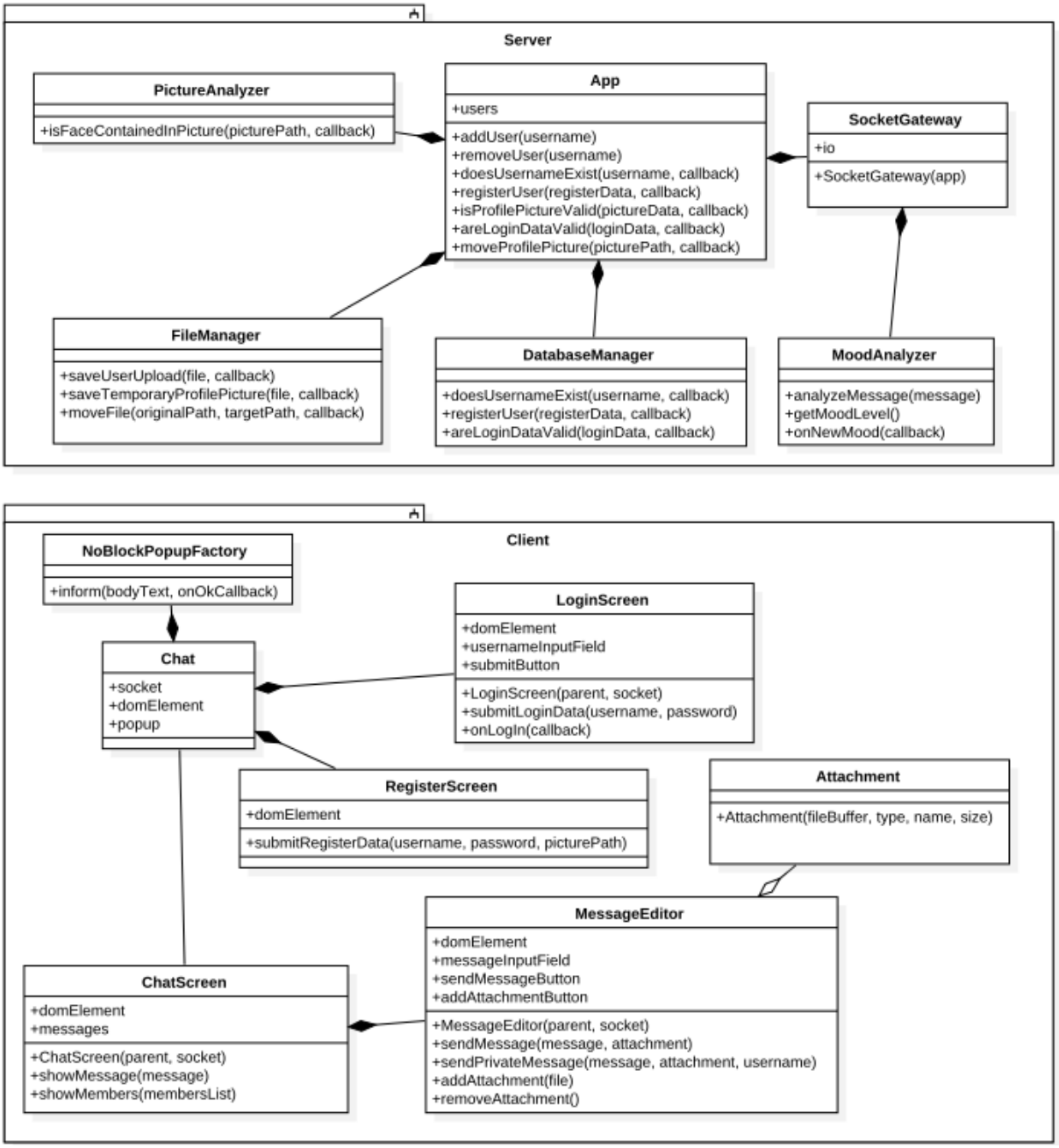


A list of the users

2.3 Component Diagram



2.4 Class Diagram



3 Results and Lessons Learned

3.1 Fulfilled and unfulfilled requirements

All the functional requirements were fulfilled.

From the non-functional ones, 1 to 3 were fulfilled.

Concerning the 4th one:

"Having deployed and tested your enhanced chat server, run a dynamic "Application Security on Cloud" test on your application and put the results in your presentation."

We run the scan and it went on for over 12 hours. The next day it produced the following error:

"Scan Failed! Our service sends a high volume of traffic when scanning, and this destabilized the site. Therefore, the scan did not complete. Please verify that the site is up and able to support a high volume of traffic before scanning again. 502 Bad Gateway: Registered endpoint failed to handle the request."

From this point our website under https://awesome-chat-app.eu-de.mybluemix.net/ produced the following error:

"502 Bad Gateway: Registered endpoint failed to handle the request."

The following measures were taken in order to remove the error:

1. Restarting the application.
2. Increasing the available RAM and restarting.
3. Deleting the application and creating a new one from the beginning, with a new toolchain. New URL: https://whatever-chat.eu-de.mybluemix.net/

Searching on the internet didn't give much information besides the 2nd point, and we didn't have much time left for further searches. Strangely enough the application works perfectly locally.

By looking the online logs, we realized that it doesn't crash, but produces the following error:

```
"whatever-chat.eu-de.mybluemix.net - [2017-12-01T08:40:28.189+0000]
"GET /favicon.ico HTTP/1.1" 502 0 67 "https://whatever-chat.eu-
de.mybluemix.net/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.94
Safari/537.36" "169.50.33.195:18274" "169.50.40.134:62737"
x_forwarded_for:"134.103.106.161" x_forwarded_proto:"https"
vcap_request_id:"49841340-d50d-4293-4109-aa75ad57fb26"
response_time:0.002116814 app_id:"bf368233-100b-41ca-8bd3-
a367865abfa7" app_index:"0" x_global_transaction_id:"2992269761"
true_client_ip:"-" x_b3_traceid:"ebab821acb390d97"
x_b3_spanid:"ebab821acb390d97" x_b3_parentspanid:"-" "
```

Unfortunately, we had no time to resolve this.

Concerning the 5th one:

"Fix at least three identified issues from requirement 4."

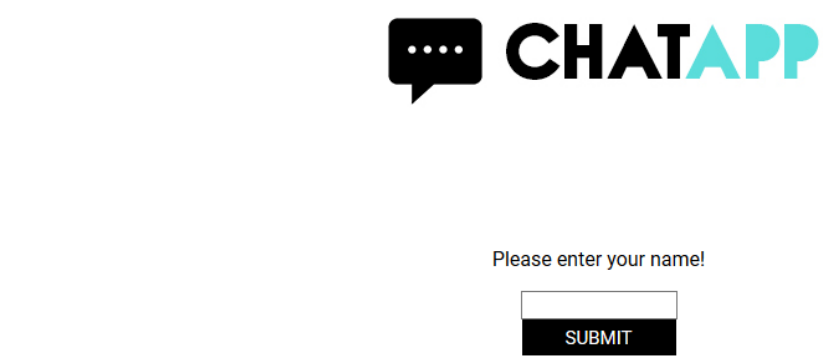
Since we couldn't even produce a report, we went through the list in the Email and tried to fix some of them.

Problem	Result
Check SRI support (Sub resource Integrity)	No time left
Header "Content-Security-Policy" missing	Should be ok
Header "X-XSS-Protection" missing	Should be ok
HSTS-Header (HTTP Strict-Transport-Security) missing	Should be ok
HTML attribute 'autocomplete' for password not deactivated	Should be ok
SHA-1 cipher suites were detected	Should be ok

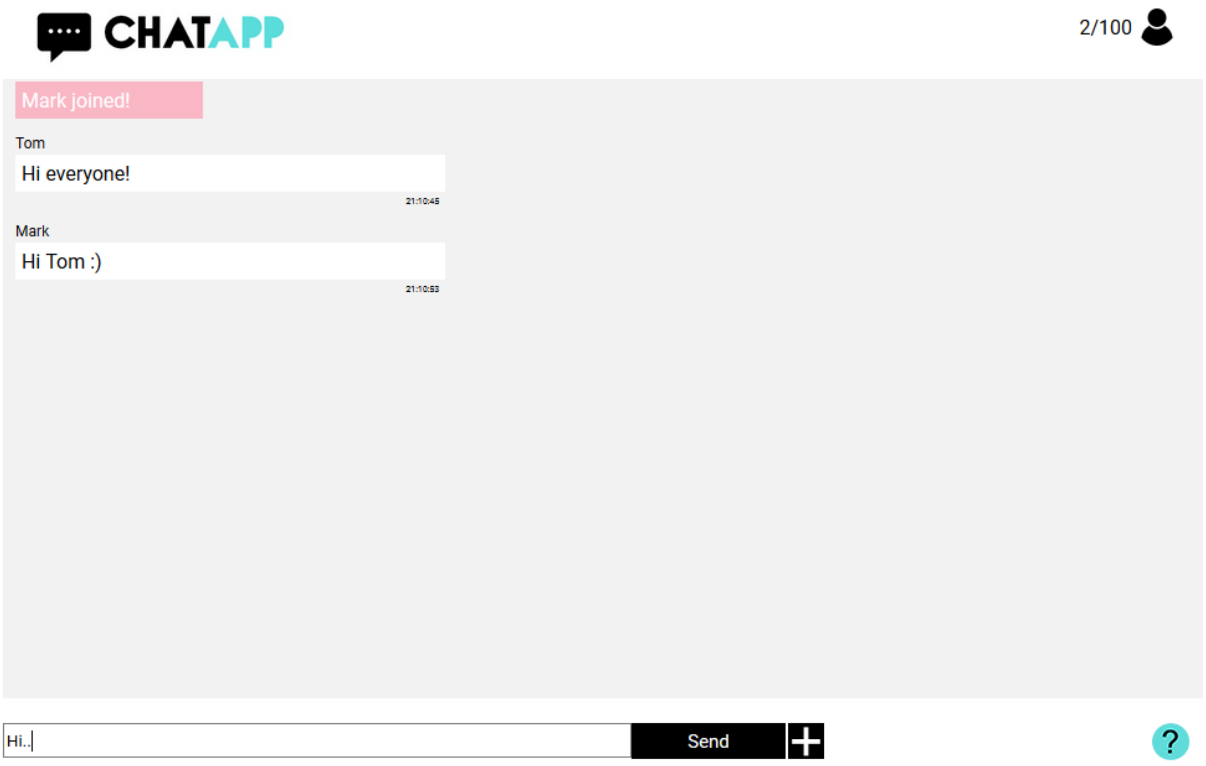
3.2 Installation and running the server

1. Go into the src folder and open a cli window
2. Enter npm install. This installs all the necessary libraries defined under package.json
3. Enter npm start. This starts the server as defined under package.json
4. Now your server is running on port 8080
5. Open a web browser and enter localhost:8080 in the address field

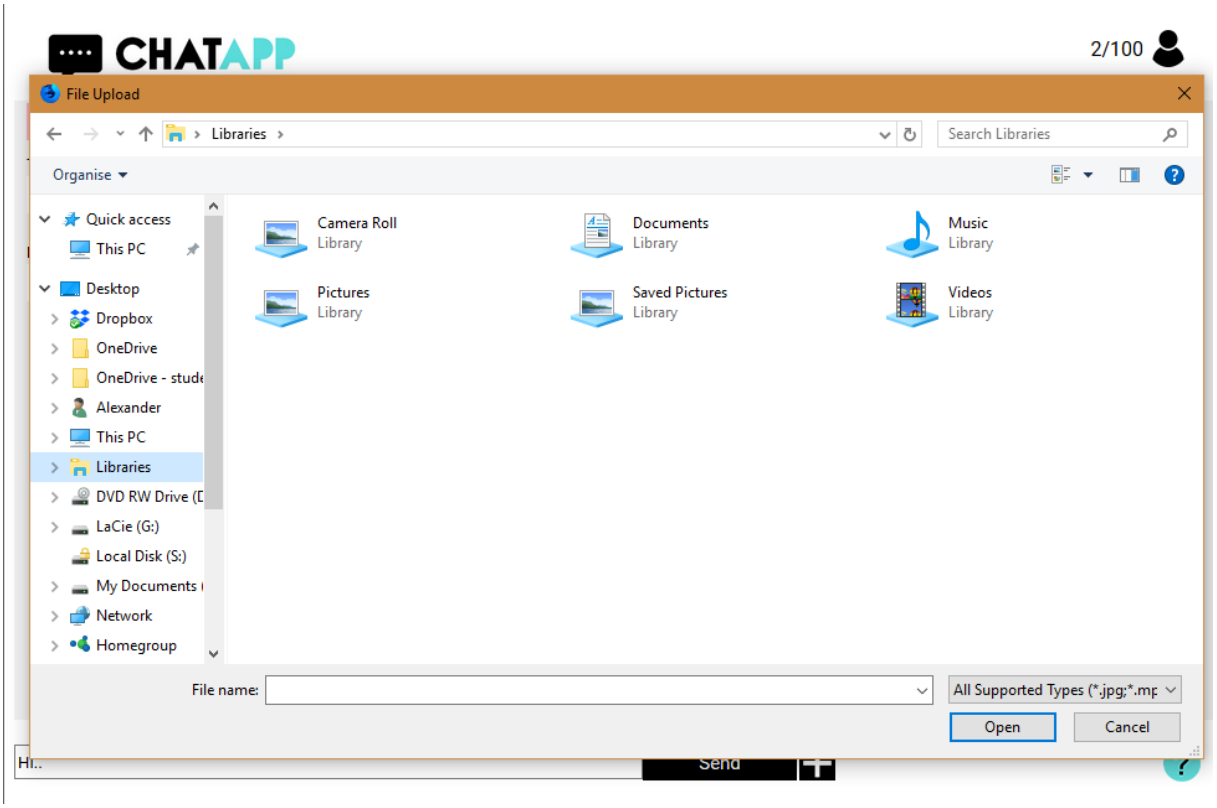
3.3 Pictures of the product



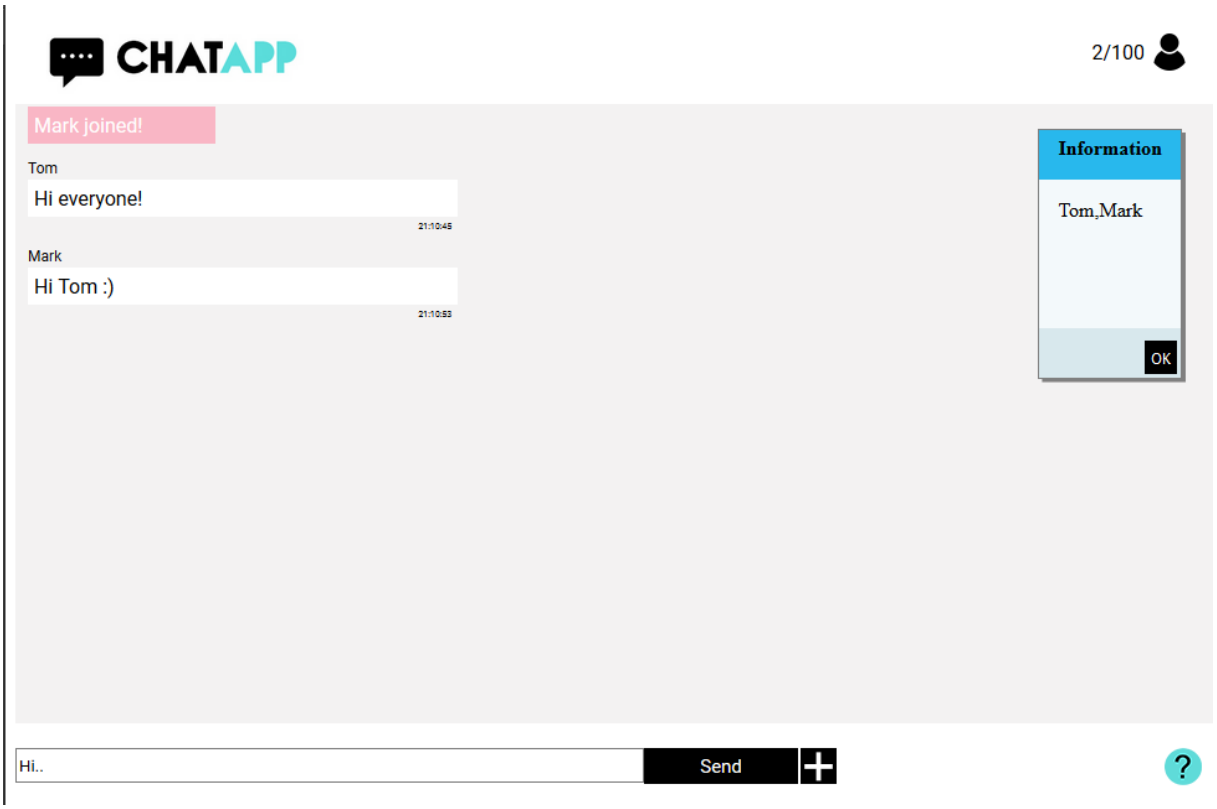
Login Screen



Chat Room Screen



The file chooser in the chat



Chat Room with All User list open

CHATAPP

Please enter your name!

Please enter your password

Please confirm your password

REGISTER

Already registered? Press here to log in

Register Screen with Visual Recognition

3.4 Using Webpack

We implemented a build process to get rid of the big number of .js files. This reduces the number of GET requests and improves loading time for the app.

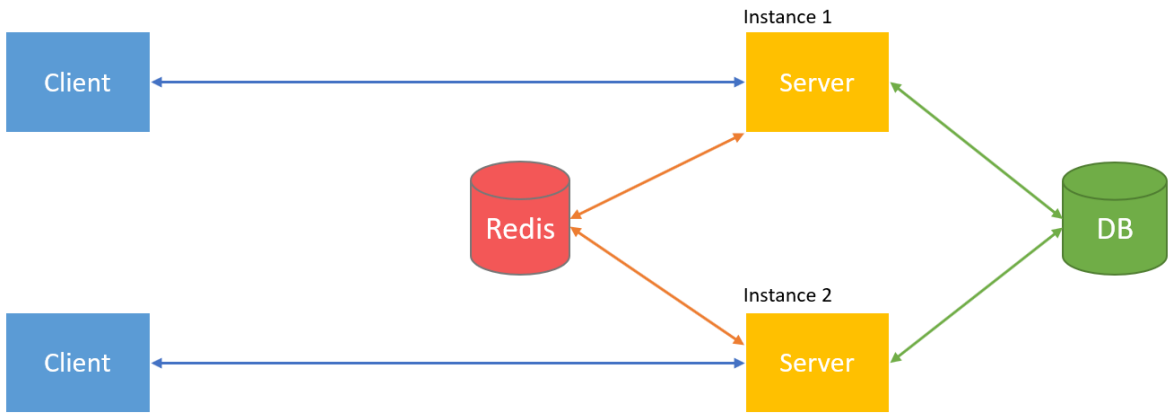
If you make a change to the frontend, run the `webpack` command on the root of the project. This will read the webpack.config.js file and build the .js files under /client/scripts into one file named bundle.js saved under /client/dist.

You can also run `npm run-script build & npm run-script start` for building and starting the server with one command.

3.5 Future improvements

A future improvement could be to implement the webpack build process into the IBM Cloud toolchain. This would save the programmer of thinking about it.

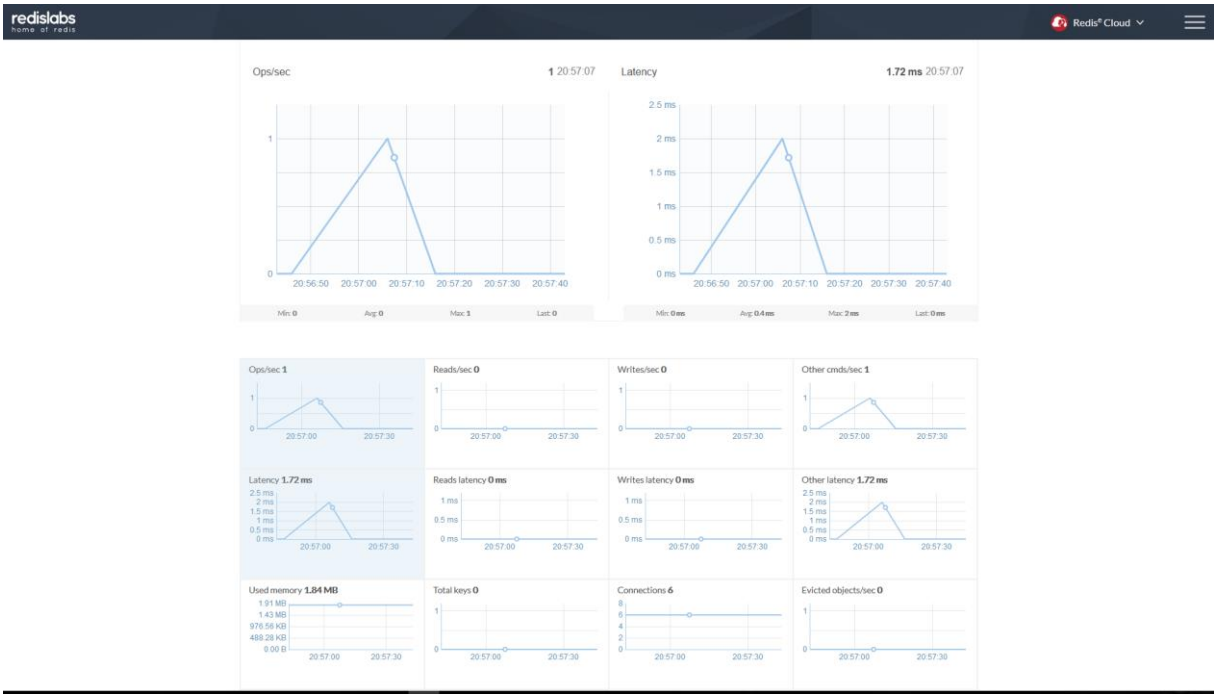
3.6 Multiple Instances



To run the application on multiple instances, we use redis, an open source key-value cache and storage system. During the activity within the application two or more instances of server need to communicate with each other to exchange data. In order to do that, redis provides temporary storage to store data. The server, after receiving a request, first checks if the cache has the response available. If so, it sends it to the client. If not, it queries the database as usual, and stores the response in the cache before sending it back to the client. This way, every response is either cached, or retrieved from the cache, and as a result, the load to our server and database is reduced.

The idea behind this approach is also to prevent a whole server being crashed after one single instance for some reason failed function.

During the implementation, we don't found any instance failure after scaling out the service. Everything works perfectly fine as expected.



Redislabs dashboard

3.7 Lesson learned

1. Learn to change a webpage using JavaScript functions without loading a new HTML document, gives a responsive feeling to the user.
2. Learn to create an application using Node JS.
3. Learn to deploy an application to IBM Bluemix.
4. Learn to use IBM Watson.
5. Learn to use Cloudant NoSQL DB.
6. Learn to make an application run on multiple instances using redis.