Chat program in Angular

In this assignment, you should write a chat program using Angular and Socket.IO. A server written in Node.js is supplied (see below), so your task is to write the client (you should not have to touch the server code).
Following is a list of functional requirements:

- (10%) The user should upon arrival specify his/her name/nick. If the nickname is free, i.e. no other user is currently active with the same name, (s)he can proceed, otherwise a new nick must be provided. Note: no password is required, just a nickname.
- (10%) After the user has identified, (s)he should see a list of chat rooms already active.
- (5%) The user should be able to join a chat room, and leave a room as well. It should of course also be possible to create a new room.
- (15%) Inside a given room, the user should be able to send messages to the room, see previous messages, and see new messages appear in real time (without having to refresh the page manually).
- (10%) It should be possible to send a private message to another user.
- (15%) The creator of a room should be able to kick a user from a room. A user which has been kicked from a room can re-enter. The creator can also ban a user, which means (s)he won't be able to join the room again.

In addition, the following technical requirements are given:

- (15%) Each component (controller, service, etc.) should be in a single file, but the files should be concatenated and minified when prepared for production.
- (10%) All external dependencies (Angular etc.) should be installed using npm or bower
- (10%) The code should go through JSHint/JSLint/TSLint without warnings. A webpack/grunt/gulp file should be included to ensure running jshint/jslint/tslint is easy. The following is an example of how jshint should be configured:

```
options: {
        curly:  true,
        immed:  true,
        newcap: true,
        noarg:  true,
        sub:    true,
        boss:   true,
        eqnull: true,
        node:   true,
        undef:  true,
        globals: {
               _:       false,
               jQuery:  false,
               angular: false,
               moment:  false,
               console: false,
               $:       false,
               io:      false
               }
        }
```

Server

A server is provided, and the source code for it can be downloaded as an attachment with the assignment. The server is written in Node.js, which must therefore be installed.
Before the server can be started, the dependencies for it must be installed by navigating to the server folder in the command line, and executing the following command:

```
npm install
```

Then, to start the server, execute the following command:

```
node chatserver.js
```

This will start a socket.io server on port 8080.

The server supports the following commands:

- **adduser**
  Your application should call this when a user joins the application, after the user has chosen a nickname.
  - Parameters:
    - the nick chosen by the user (string)
    - a callback function, which accepts a single boolean parameter, stating if the username is available or not.
      Example:
      ```
      socket.emit("adduser", "dabs", function(available){
              if (available){
              // The "dabs" username is not taken!
              }
      });
      ```
- **rooms**
  Should get called to receive a list of available rooms.
  There are no parameters.
  The server responds by emitting the "roomlist" event (the client needs to listen to this event from the server).
- **joinroom**
  Should get called when a user wants to join a room. Note that the API supports a password-protected room, however this is optional, i.e. your implementation doesn't have to support this.
  - Parameters:
    - an object containing the following properties: { room: "the id of the room, undefined if the user is creating a new room", pass: "a room password - not required"}
    - a callback function which accepts two parameters: a boolean parameter, stating if the request was successful or not. and if not (due to password protection or because of something else), the reason why the join wasn't successful.

  The server responds by emitting the following events: "updateusers" (to all participants in the room), "updatetopic" (to the newly joined user, not required to handle this), "servermessage" with the first parameter set to "join" (to all participants in the room, informing about the newly added user). If a new room is being created, the message "updatechat" is also emitted.
- **sendmsg**
  Should get called when a user wants to send a message to a room.
  - Parameters:
    - a single object containing the following properties: {roomName: "the room identifier", msg: "The message itself, only the first 200 chars are considered valid" }

    The server will then emit the "updatechat" event, after the message has been accepted.
- **privatemsg**
  Used if the user wants to send a private message to another user.
  - Parameters:
    - an object containing the following properties: {nick: "the userid which the message should be sent to", message: "The message itself" }
    - a callback function, accepting a single boolean parameter, stating if the message could be sent or not.

  The server will then emit the "recv_privatemsg" event to the user which should receive the message.

- **partroom**
  Used when a user wants to leave a room.
  - Parameters:
    - a single string, i.e. the ID of the room which the user is leaving.

  The server will then emit the "updateusers" event to the remaining users in the room, and a "servermessage" with the first parameter set to "part".
- **disconnect**
  Used when a user leaves the chat application.
  There are no parameters.
  The server will emit the following events: "updateusers" to each room the user had joined (and hadn't explicitly left), "servermessage" with the first parameter set to "quit".
- **kick**
  When a room creator wants to kick a user from the room.
  - Parameters:
    - an object containing the following properties: { user : "The username of the user being kicked", room: "The ID of the room"
    - a callback function, accepting a single boolean parameter, stating if the user could be kicked or not.

  The server will emit the following events if the user was successfully kicked: "kicked" to the user being kicked, and "updateusers" to the rest of the users in the room.
- **ban**
  Allows an operator to ban another user from a room.
  - Parameters:
    - an object containing the following properties: { user : "The username of the user being banned", room: "The ID of the room"
    - a callback function, accepting a single boolean parameter, stating if the user could be banned or not.

  The server will emit the following events if the user was successfully banned: "banned" to the user being banned, and "updateusers" to the rest of the users in the room.

- **users**
  This should get called to get a list of all connected users.
  There are no parameters for this function.
  The server will emit the "userlist" event back to the caller, containing a list of userids currently "logged in"

Note that the following commands are also supported, but it is not required to use them (unless you want to write a really awesome chat application). Feel free to experiment with these features.

- **op**
  Allows the creator of a room to convert another user to a "op", i.e. grant that user the same priviledges as the creator of the room.
- **deop**
  Allows an op to "deop" another operator. Note: there is nothing that prevents the following events from happening: A creates a room, B joins, A ops B, B deops A.
- **setpassword**
  Sets a password for a room. Only an OP can perform this action.
- **removepassword**
  Reverts the "setpassword" command.
- **settopic**
  Sets a "topic" for a room. Only an OP can perform this action.
- **unban**
  Reverts the "ban" operation.

Handin

Hand in a single archive file, containing all the source files for the application, including instructions on how to install and run the application in a README.md file. Note: unless you make changes to the supplied server, then you DON'T need to hand in that code! As a matter of fact, your code should absolutely not be integrated into that folder, the server and the client should be stored in two separate folders!
Do note that your solution should NOT include the bower and/or the node_modules folders!