Overview

This basic chat functionality uses node, express, mysql, socket.io and passport.js

clone: `https://github.com/ubersensei/chat.git`
go to: `lib/db/dbInitialize.js` and update ***your*** `database,user,password`

$ `sudo npm install`
$ `npm start`
direct browser to `http://localhost:3000/`

On `npm start`

- The **database**, **users** table and **messages** table are dropped and created
- 3 users are inserted: **user1**, **user2**, **user3** (all have same password '*secret*')
- 6 messages are inserted: one message from each user to the other two users

Try out the below test cases manually (using UI, manual logging etc.) - it works as expected.

// TODO: automated testing (mocha, chai …)

(preferably, use real environments/requests as opposed to fake e.g. simulated XHRs)

signin - http

- send http signin request (submit signin form with name/password)
- if username is missing, get flash message ('*Missing credentials*')
- if database table doesn't exist, get a 404 error
- if successfully signed in,
    - get data on messages involving self
    - trigger socket.io at client and establish connection
        - update user's `socket.id` in the database

socket messaging

- choose an ***offline*** target user (say, user2) and send chat message
- when message reaches server,
    - try to store this message in db,
        - if successful, confirm delivery_status = 'delivered'
        - if fail, confirm delivery_status = 'not delivered'
- login as user2 (and now user2 is online i.e. has a socket.id)
- when message reaches server and if target user's socket.id exists, then
    - try to store this message in db,
        - if successful, confirm delivery_status = 'seen' to sender
        - if fail (e.g. database table doesn't exist), confirm delivery_status = 'not delivered' to sender

High level code walk-through

signin - http

Fairly straight forward in that the signin form does a `POST` to /login, which is handled from within routes/auth.js. Passport.js functions do their magic, and signin happens.

Note that clientJS.js is loaded ***only*** after successful signin.
clientJS.js then fires up ajax functions (see/scroll to the bottom of this file) to get data related to users(self and others) and messages. This data is stored at the client side (and also rendered on the UI).

clientJS.js also initiatializes socket.io functions.
When socket.io connection is established, the signedin user's socket.id is broadcast to all other (online) users. See lib/socketFunctions.js

socket messaging

when a user sends a message, this message content is immediately rendered on the sender's UI. However, the delivery_status of the message is updated only after confirmation from the server via a callback mechanism.