Graded Project “Web Programming”

In the graded project for "Web Programming", you are asked to develop a Twitter-like chat application that provides a communication channel associated with BINFO internships. Channels will carry messages (“tweeds”) related with a dedicated internship, or among all students and supervisors associated with internships in the current semester. More precisely, the following functionalities must be realized by your system:

1. A super user (account: "root", initial password "admin") can manage the internships available in the system (create new internships, modify internship information like “title”, “student”, “local supervisor”, "academic supervisor", or modify / delete existing information). He can also create or disable non-admin users ((i.e. students or lecturers) are never deleted, but only disabled). All non-admin users must have a secret password created during account generation and stored (for simplicity as plaintext) in the DB. The super user can only do these actions after having logged-in successfully onto the server. All activities of the super user are done on the server side, implemented with either PHP or JS / Node.js and using a MySQL DB.

2. All activities of non-admin users are completely implemented in client side JavaScript, using RESTful web services for interaction with the server side (with JSON data encoding). The web service providers on server side can be implemented with either PHP or JS / Node.js.

3. A non-admin user has to login before he can use the software. For login, the following protocol must be used:
   (a) User sends initial message "request login" with some random secret character string x of at least 16 byte to server WS.
   (b) Server WS responds with server secret d (which might be the current date and time).
   (c) Client sends second message as SHA256(x + pwd + d), where "+" is string concatenation, pwd is the user's secret password, and SHA256 is a cryptographic hash function – you can use an existing appropriate JS module providing this algorithm.
   (d) Server checks correctness of provided string using the password information in the DB; in case of success, responds with a sufficiently long random SESSIONID.
   (e) In all future requests, this SESSIONID must be included with the request – this id provides a way for the server to identify the current session and client (in practice, all connections should of course use HTTPS for traffic encryption; for simplicity, only HTTP is required in our example such that this SESSIONID is not really securely transmitted).
   (f) At a later time, the user can send a logout message together with the SESSIONID, which forces the server to delete the SESSIONID from the set of valid sessions.

4. After login, an ordinary user can register to channels for his internship available on the

server. Users can tweet a message onto the internship channel, which is then transmitted via a REST WS to the server. The server uses web sockets to push that message to all logged in clients currently registered for the particular channel. Clients will show received messages immediately. All messages should have a message id, and you can add some meta information to messages to allow operations as "reply-to" for some previous message, output message sender, etc.

Submission Rules: The application must be developed with the Docker application provided – use either the NGINX-PHP-MySQL or the NGINX-Node.JS-MySQL applications provided with possible adaptations if required. **Archive the complete Docker installation directory** including all sources and required code for used frameworks, possible adaptions of the Docker setup files, and a MySQL textual DB dump **into a single zip archive** (rar archives are also allowed), and **upload that single archive file to Moodle**. Make sure that the application can be build and run within Docker without further manipulations from my side. If the size of this zip archive exceeds the Moodle upload limit, then please put the archive on a public file server of your choice (Dropbox or an alternative), and upload to Moodle only a file containing the download link (must be valid until end of January 2020).

Please pre-define for my testing a channel "Internship1" and two related users (supervisor "vm" with password "vm", and "student1" with password "student1"). Try to make your application as user-friendly as possible. Give some explanation on your implementation (what is working, what part is maybe not working, what I should know to test the system) in a textual README file.

Bonus:

1. All messages can be stored on the server side. The client should have the possibility to retrieve a tweet history for a specific time period provided in the request. If no such period is provided, then all tweets in that channel should be returned and displayed. Implementing this feature gives a bonus of **3 Points**.
2. Using a JS framework for the client side code (ember.js, react.js, or angular.js) will lead to an additional bonus of **3 points**. Please list in the textual README file all the frameworks that you are using.

Deadline: 5th January 2020, 23:59. Late submissions will lead to a reduction of 1.5 points per day.