COMP3322B Modern Technologies on World Wide Web

Assignment 2 (20%)

[Learning Outcomes 2, 3]

Due by: 23:55, Wednesday May 8 2019

Overview

In this assignment, we are going to develop a simple single-page social network application using the MERN stack (MongoDB, Express.JS, ReactJS, and Node.js). The main workflow of the social network application is as follows.

- Upon loading, the sketch of the page is shown in Fig. 1:

Fig. 1

- After a user (e.g., Andy) has logged in, the sketch of the page is in Fig. 2. Names of a list of starred friends are shown on the left, and the posts by all friends and associated comments are displayed on the right.

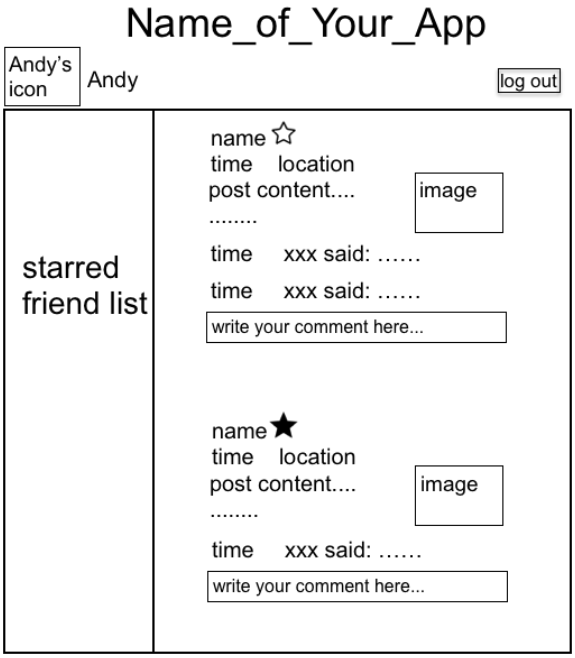

Fig. 2

- After you have entered a comment in a comment input textbox (e.g., in the second textbox in Fig. 2) and pressed enter, the comment is displayed above the textbox (e.g., Fig. 3).

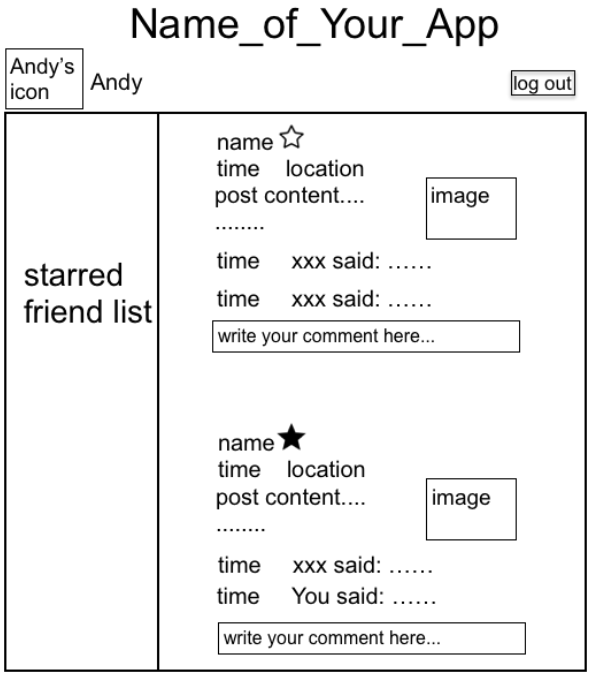

Fig. 3

- On a page view as in Fig. 3, when you double click a comment you have posted, e.g., on the line shown "Time You said: xxx", a popup box will be shown, asking you to confirm deletion of the post or not (Fig. 4). If you confirm deletion by clicking "OK" button, the comment is deleted and the page view goes back to Fig. 2; otherwise, if you click "Cancel", the page view remains as Fig. 3.

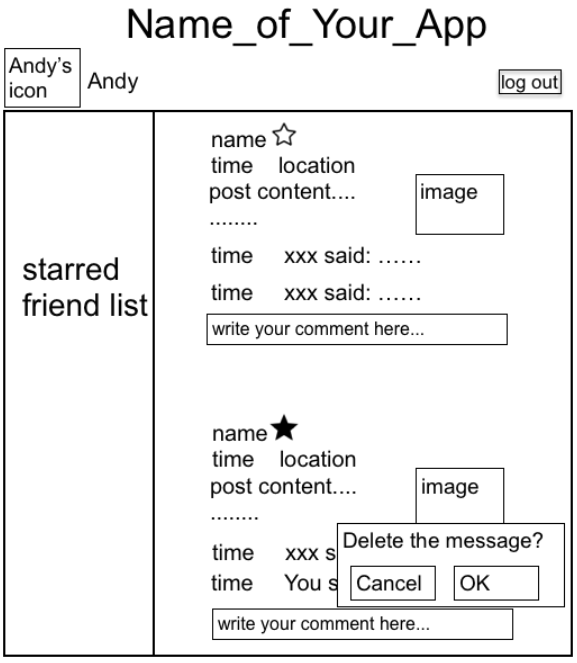

Fig. 4

star a friend

- Within the posts division on the right, there is a star icon following the name of the friend in each post. There are two versions of the star icon: one to display when the friend is not starred (e.g., the star icon following the first name in Figures 2-4), and one to display when the friend is starred (e.g., the star icon following the second name in Figures 2-4). When you click on a star icon, it will be switching between the two versions, indicating switching between starred and un-starred states of the respective friend. For example, after you click the first start icon on Fig. 2, the page view becomes one in Fig. 5. The list on the left shows the names of starred friends, and should be updated accordingly.

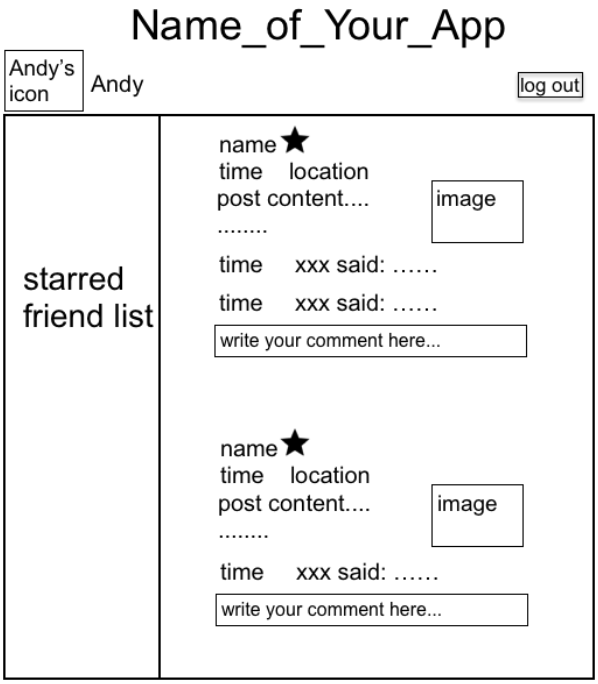

Fig. 5

- After you click the area containing Andy's icon and name at the top of the page, a profile update page is shown on the right (Fig. 6). Mobile number, home number and address of the current user are displayed in the three input textboxes. You can edit the information and then after you click the "Save" button, the page view goes back to one as shown in Fig. 2, and the information of the current user is updated to the server side.

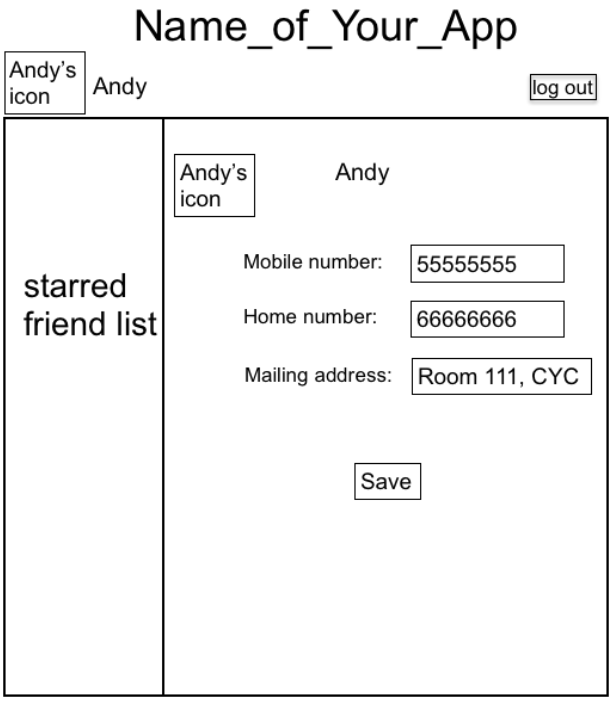

Fig. 6

- The web page should periodically retrieve new comments and comment deletions from friends on the displayed posts from the backend web service, and update the page display accordingly, if it is showing a page view as in Fig. 2, 3, 4 or 5.

- When you click the "log out" button, the page view goes back to Fig. 1.

We are going to achieve this application by implementing code in a backend Express app and a frontend React app.

- Express app:
  app.js
  ./routes/socialservice.js

- React app:
  ./src/App.js
  ./src/index.js
  ./src/App.css

Task 1. Backend Web Service

We implement the backend web service logic using Express.js. The web service is accessed at http://localhost:3001/xx.

Preparations

1. Following steps in Lab 6, install the Node.js environment and the Express framework,

create an Express project named **SocialService**, add dependencies for MongoDB in **package.json,** and install dependencies.

2. Following steps in Lab 6, install MongoDB, run MongoDB server, and create a database "assignment2" in the database server.

Insert a number of user records to a userList collection in the database in the format as follows. We will assume that user names are all different in this application. Here friendId is the value of _id generated by MongoDB in the userList collection for the respective user. You can find out such a _id using db.userList.find().

```
db.userList.insert({'name': 'Andy', 'password': '123456', 'icon': 'icons/andy.jpg', 'mobileNumber': '55555555', 'homeNumber': '66666666', 'address': 'Room 111, CYC', 'friends':[{'friendId': 'xxx', 'starredOrNot': 'N'}, {'friendId': 'xxx', 'starredOrNot': 'Y'}], 'lastCommentRetrievalTime':'22:00 Sat Apr 13 2019'})
```

Create a folder "icons" under the public folder in your project directory. Copy a few icon images to the icons folder. For implementation simplicity, in this assignment, we do not store icon images in MongoDB. Instead, we store them in the harddisk under the ./public/icons/ folder, and store the path of an icon in the userList collection only, using which we can identify the icon image in the icons folder.

Insert a few records to a postList collection in the database in the format as follows, each corresponding to one post in the social app. Here userId should be the value of _id of the record in the userList collection, corresponding to the user who posted the post.

```
db.postList.insert({'userId': 'xxx', 'time': '20:32 Sat Apr 13 2019', 'location': 'Hong Kong', 'content': 'post content'})
```

Insert a few records to a commentList collection in the database in the format as follows, each corresponding to one comment in the social app. Here postId should be the value of _id of the respective post record in the postList collection, which the comment is posted on; userId is the _id of the user in the userList collection, who posted the comment.

```
db.commentList.insert({'postId': 'xxx', 'userId': 'xxx', 'postTime': '21:00 Sat Apr 13 2019', 'comment': 'comment content', 'deleteTime': ''})
```

Implement backend web service logic (SocialService/app.js, SocialService/routes/socialservice.js)

app.js **(7 marks)**

In **app.js**, load the router module implemented in ./routes/socialservice.js. Then add the

middleware to specify that all requests for http://localhost:3001/ should be handled by this router.

Add necessary code for loading the MongoDB database you have created, creating an instance of the database, and passing the database instance for usage of all middlewares.

Also load any other modules and add other middlewares which you may need to implement this application.

We will let the server run on the port 3001 and launch the Express app using command "node app.js".

./routes/socialservice.js **(35 marks)**

In **socialservice.js**, implement the router module with middlewares to handle the following endpoints:

1. **HTTP POST requests for** http://localhost:3001/signin. The middleware should parse the body of the HTTP POST request and extract the username and password carried in request body. Then it checks whether the username and password match any record in the userList collection in the database. If no, send "Login failure" in the body of the response message. If yes, create a cookie variable "userId" and store this user's _id in the cookie variable (do not set expiration time of the cookie, such that it expires when the user's browser is closed), and do the following:

(1) retrieve name and icon of the current user (according to the value of the "userId" cookie), _id, name, icon and starredOrNot status of friends of the current user, information of posts (_id, time, location, content) posted by friends of the current user, and information of comments (_id, name, postTime, content) submitted by friends on those posts from the respective collections in the MongoDB database. [The comment records retrieved should have an empty deleteTime (''), i.e., those not deleted.]

(2) update lastCommentRetrievalTime of the user's record in the userList collection according to the current time.

(3) send all retrieved information in (1) as a JSON string to the client if database operations are successful, and the error message if failure. You should decide the format of the JSON string and parse it accordingly in the front-end code to be implemented in Task 2.

2. **HTTP GET requests for** http://localhost:3001/logout. The middleware should unset the "userId" cookie variable, clear lastCommentRetrievalTime in the user's record in the userList collection (e.g., set its value to ''), and send an empty string back to the user.

3. **HTTP GET requests for** http://localhost:3001/getuserprofile. Retrieve mobileNumber, homeNumber and address of the user from the userList collection based on the value of "userId" cookie. Send retrieved information as a JSON string in the body of the response message if database operations are successful, and the error message if failure. You should

decide the format of the JSON string to be included in the response body.

4. **HTTP PUT requests for** http://localhost:3001/saveuserprofile. The middleware should update the mobileNumber, homeNumber and address of the user in the userList collection based on the value of "userId" cookie and the data contained in the body of the request message. Return an empty string to the client if success and the error message if failure.

5. **HTTP GET requests for** http://localhost:3001/updatestar/:friendid. The middleware should update the starredOrNot of the friend (decided by friendid carried in the URL) of the user in the userList collection, based on the value of "userId" cookie. If the current value of starredOrNot is Y (N), update it to N (Y). Return an empty string to the client if success and the error message if failure.

6. **HTTP POST requests for** http://localhost:3001/postcomment/:postid. The middleware should parse the body of the HTTP POST request and extract the comment carried in the request body. Then it saves the comment into the commentList collection together with the postid carried in the URL, the poster's _id (based on the value of "userId" cookie), and the current time on the server as the postTime, with the deleteTime set to ''. Return an empty string to the client if success and the error message if failure.

7. **HTTP DELETE requests for** http://localhost:3001/deletecomment/:commentid. In the middleware, instead of completely deleting the comment record (corresponding to the commentid in the URL) from the commentList collection, we update deleteTime in the comment record to the current time on the server. In this way, we tell that a comment in the commentList collection has been deleted or not, according to whether the deleteTime is not empty or empty (this is to facilitate our implementation of middleware 8 below, which sends only changes in the commentList collection to the client side). Return an empty string to the client if success and the error message if failure.

8. **HTTP GET requests for** http://localhost:3001/loadcommentupdates. Retrieve information of new comments submitted by friends on the posts from friends of the current user (decided by the value of the _id cookie) from the commentList collection, which were posted after the lastCommentRetrievalTime of the current user's record in the userList collection (i.e., those comments that the current user has not retrieved from the web service yet); also retrieve _Id's of comments (submitted by friends on the posts from friends of the current user) which were deleted after lastCommentRetrievalTime of the current user (i.e., those comment deletions that the current user has not been aware of). Then update the lastCommentRetrievalTime in the current user's record in the userList collection according to the current time. Send information of retrieved new posts and _Id's of newly deleted comments as a JSON string in the body of the response message if database operations are successful, and the error message if failure. You should decide the format of the JSON string to be included in the response body.

Task 2 Front-end React App

Implement the front-end as a React application. The application is accessed at http://localhost:3000/.

Preparations

Following steps in Lab 8, create a React app named **MyApp** and install the jQuery module in the React app.

Implement the React app (MyApp/src/index.js, MyApp/src/App.js, MyApp/src/App.css)

index.js **(3 marks)**

Modify the generated **Index.js** in the ./src folder of your react app directory, such that it renders the component you create in **App.js** in the "root" division in the default index.html.

App.js **(40 marks)**

App.js should import the jQuery module and link to the style sheet **App.css**.

Design and implement the component structure in **App.js**, such that the front-end page views and functionalities as illustrated in Figures 1-6 can be achieved.

Hints:

- You can decide the name of your app to be shown on the top of the page. The posts and comments should be displayed according to chronological order.

- You can use conditional rendering to decide if the page view in Fig. 1 or Fig. 2 should be rendered. Suppose the *root* component you are creating in **App.js** is FrontApp. In FrontApp, you may use a state variable to indicate if a cookie has been set for the current user (i.e., the user has logged in) or not, and then render the component presenting the respective page view accordingly. Initialize the state variable to indicate that no cookie has been set, and update it when the user has successfully logged in and logged out, respectively.

- In the component implementing the page view as in Fig. 1, add an event handler for the onClick event on the "Sign in" button (consider the event handler code should be implemented in the current component, or the *root* component). When handling the event, send an HTTP POST request for http://localhost:3001/signin (refer to this website for AJAX cross-origin with cookies: http://promincproductions.com/blog/cross-domain-ajax-request-cookies-cors/).

According to the response received from the backend service, remain on the page view and display the "Login failure" message at the top of the page, or render the page view as in Fig. 2.

- When handling the onClick event on the "log out" button in a page view as in Figures 2-6, send an HTTP GET request for http://localhost:3001/logout and handle the response accordingly.

- When handling the onClick event on the area containing the user's icon and name in a page view as in Figures 2-6, send an HTTP GET request for http://localhost:3001/getuserprofile, and render a page view as in Fig. 6. Again, you may use a state variable in the component representing the right division in the page view of Figures 2-6, to indicate whether the content as in Fig. 6, or content as in Figures 2-5 should be displayed in the right division.

- When handling the onClick event on the "Save" button in a page view as in Fig. 6, send an HTTP PUT request for http://localhost:3001/saveuserprofile and handle the response accordingly.

- When handling the onClick event on a star icon in a page view as in Figures 2-5, send an HTTP GET request for http://localhost:3001/updatestar/xxx (where xxx is friendId). If success response is received, update the star icons displayed with all the posts from the same friend and the starred friend list accordingly.

- You can include the input textbox underneath each post in a page view as in Figures 2-5 in a <form> element, such that the onSubmit event on the form is triggered when "enter" is pressed in the input textbox. When handling the onSubmit event, send an HTTP POST request for http://localhost:3001/postcomment/xxx (where xxx should be the postId). When success response is received, display the newly entered comment above the textbox.

- When handling the onDoubleClick event on a displayed comment of the current user in a page view as in Figures 2, 3, 5, show a confirmation box as in Fig. 4; when deletion is confirmed, send an HTTP DELETE request for http://localhost:3001/deletecomment/xxx (where xx is the commentId). When success response is received, remove the comment from the page view.

- To implement periodical retrieval of comment updates on the posts, we create a timer using the setInterval method. Refer to the examples on https://www.w3schools.com/jsref/met_win_setinterval.asp and https://reactjs.org/docs/state-and-lifecycle.html. Especially, in the component that you create to render the right division in a page review as in Figures 2-5, implement its componentDidMount() function by creating a timer in there; set the timeout interval to be a few seconds. In the timeout function called at each timeout interval,

send an HTTP GET request for http://localhost:3001/loadcommentupdates, and upon success response, display information of new comments received and remove comments based on the received _Id's of newly deleted comments in the page view. Implement the componentWillUnmount() function of the component as well, to stop the timer using clearInterval (https://www.w3schools.com/jsref/met_win_clearinterval.asp).

App.css **(10 marks)**
Style your page views nicely using CSS rules in **App.css.**

Other marking criteria:

(**5 marks**) Good programming style (avoid redundant code, easy to understand and maintain). You are encouraged to provide a **readme.txt** file to let us know more about your programs.

Submission:

You should zip the following files (in the indicated directory structure) into a **yourstudentID-a2.zip** file

SocialService/app.js
SocialService /routes/socialservice.js
MyApp/src/App.js
MyApp/src/index.js
MyApp/src/App.css

and submit the zip file on Moodle:
(1) Login Moodle.
(2) Find "Assignments" in the left column and click "Assignment 2".
(3) Click "Add submission", browse your .zip file and save it. Done.
(4) You will receive an automatic confirmation email, if the submission was successful.
(5) You can "Edit submission" to your already submitted file, but ONLY before the deadline.