COMP3322A Modern Technologies on World Wide Web

Assignment 2 (22%)

[Learning Outcome 2]

Due by: 23:55, Friday December 16 2022

Overview

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

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

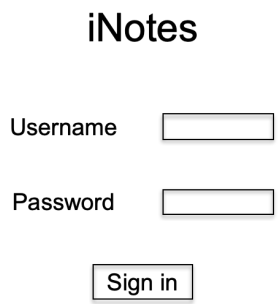

Fig. 1

- After a user (e.g., Andy) has logged in, the sketch of the page is in Fig. 2. The user's icon, user name and a logout button are displayed on the top. A list of this user's notes are shown on the left panel (note title only) together with a search bar, and the right panel is empty except an icon indicating the creation of a new note (i.e., the "New note icon"). The note titles in the left panel should be listed in reverse chronological order of the last saved time of the notes. The total number of notes should be given in the ( ) on the top of the list.

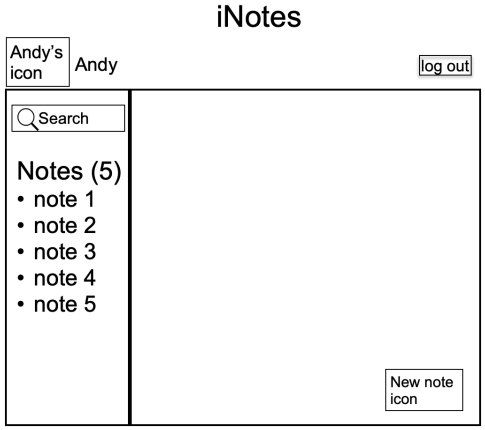

Fig. 2

- After one clicks on the “New note icon” (on any page view where it is shown), a new note creation panel shows in the right panel (Fig. 3). There is a note title input field and a note content input field, into which the user can enter texts. There is a “Cancel” button, clicking which a confirmation box “Are you sure to quit editing the note?” will be popped up: if the user confirms quitting, the page view goes back to the one shown in Fig. 2; otherwise, the current page view remains. There is a “Save” button, clicking which the newly created note is shown on the right panel, with the “Last saved” time and a “Delete” button displayed on top of the note, as shown in Fig. 4; besides, the note title should be listed in the left panel, as the first in the list (as it is the latest), the note title should be highlighted in a different color than the rest of the note titles in the list (since this note’s content is shown in the right panel), and the total number of notes in ( ) on top of the list should be incremented.

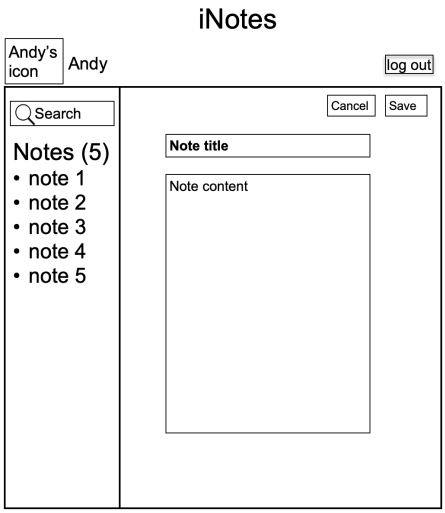

Fig. 3

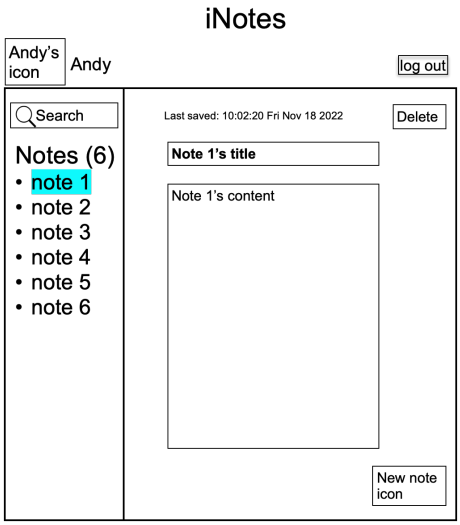

Fig. 4

- At any time, when one clicks on one note title in the left panel, the note’s content should be displayed in the right panel, as shown in Fig. 5 (which is in fact the same page view as Fig. 4), and the note title in the left panel should be highlighted. On the page view (i.e., Fig. 4 or Fig. 5’s page view), if one clicks into the note title input field or note content input field, the page view changes to the one in Fig. 6, with a “Cancel” button and a “Save” button (indicating a note editing mode). When “Cancel” is clicked, a confirmation box “Are you sure to quit editing the note?” will be popped up: if the user confirms quitting, the page goes back to the previous note view (Fig. 4 or Fig. 5); otherwise, the current page view remains. When “Save” is clicked, a page view as in Fig. 4 or Fig. 5 is shown, except that the “Last saved” time on the top of the right panel should be the updated latest note saved time.

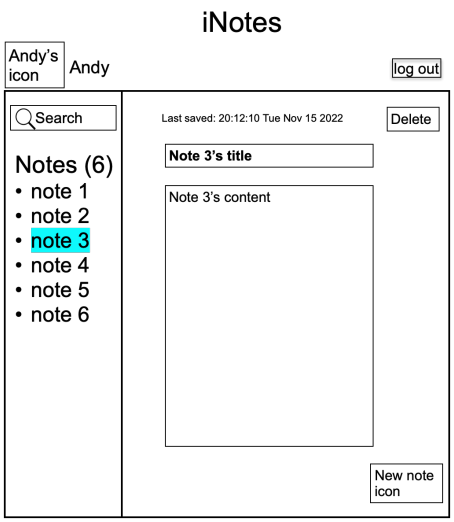

Fig. 5

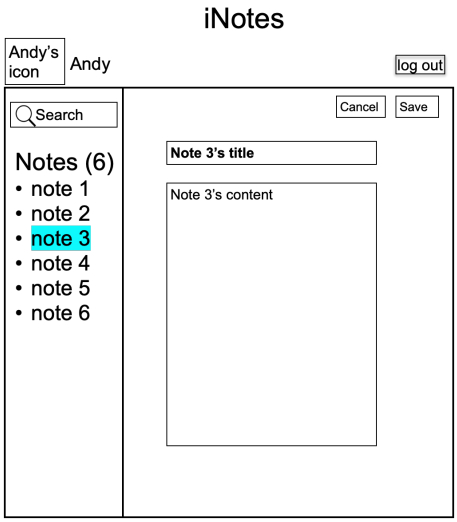

Fig. 6

- One can input a search string into the search bar at the top of the left panel and press "Enter" on the keyboard. Then only the notes whose title or content contains the search string will be listed in the left panel, ordered in reserve chronological order of their last saved time, and the number in ( ) shows the number of matched notes. On a page view as in Fig. 3, 4, 5 or 6, the search does not influence the display in the right panel, and if the note whose details are displayed in the right panel matches the search, its title in the searched list in the left panel should be highlighted. For a search on a page view as in Fig. 6, last saved title and content of the note that is being edited are used for matching the search; when the note is saved, if its title and content do not match the search, it will not be displayed in the searched list in the left panel.

- On a page view as in Fig. 4 or Fig. 5, after one clicks the "Delete" button, a confirmation box pops up showing "Confirm to delete this note?" If the user confirms the deletion, the note information will be removed from both the left and right panels. In the left panel, the total note number will be decremented; the right panel will show no note information as in Fig. 2. If the user cancels the deletion, the page view remains unchanged.

- When one clicks the "log out" button on a page view as in Fig. 2, 4 or 5, the page view directly goes back to Fig. 1. When one clicks the "log out" button on a page view as in Fig. 3 or 6, an alert box "Are you sure to quit editing the note and log out?" will be popped up: if the user confirms quitting, the page view goes back to Fig. 1; otherwise, the current page view remains.

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

- Express app:
  app.js

./routes/notes.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 **setup_nodejs_runtime_and_examples_1.pdf**, create an Express project named **NoteService**.

2. Following steps in **AJAX_JSON_MongoDB_setup_and_examples.pdf**, run MongoDB server, and create a database "assignment2" in the database server.

3. Insert a few 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.

```
db.userList.insert({'name': 'Andy', 'password': '123456', 'icon': 'icons/andy.jpg'})
```

Create a folder "icons" under the public folder in your Express project directory (NoteService). Copy a few icon images to the icons folder. For implementation simplicity, we do not store icon images in MongoDB. Instead, we store them in the harddisk under the NoteService/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 number of records to a noteList collection in the database in the format as follows, each corresponding to one note in the app. Here userId should be the value of _id of the record in the userList collection, corresponding to the user who added the note.

```
db.noteList.insert({'userId': 'xxx', 'lastsavedtime': '20:12:10 Tue Nov 15 2022', 'title': 'assigment2', 'content': 'an iNotes app based on react'})
```

Implement backend web service logic (NoteService/app.js,

NoteService/routes/notes.js)

app.js **(10 marks)**

In **app.js**, load the router module implemented in ./routes/notes.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/notes.js **(22 marks)**

In **notes.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 session variable "userId" and store this user's _id in the session variable. Retrieve name and icon of the current user (according to the value of the "userId" session variable), _id, lastsavedtime and title of all notes of the current user from the respective collections in the MongoDB database. Send all retrieved information as a JSON string to the client if database operations are successful, and the error 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 clear the "userId" session variable and send an empty string back to the user.

3. **HTTP GET requests for** http://localhost:3001/getnote?noteid=xx. Retrieve _id, lastsavedtime, title and content of the note from the noteList collection based on the value of "noteid" carried in the URL. Send retrieved information as a JSON string in the body of the response message if database operations are successful, and the error if failure. You should decide the format of the JSON string to be included in the response body.

4. **HTTP POST requests for** http://localhost:3001/addnote. The middleware should parse the body of the HTTP POST request and extract the note title and content carried in the request body. Then it saves the new note into the noteList collection together with the _id of the current user (based on the value of "userId" session variable) and the current time on the

server as the lastsavedtime. Return the lastsavedtime and _id of the note document in the noteList collection to the client in JSON if database operations are successful, and the error if failure.

5. **HTTP PUT requests for** http://localhost:3001/savenote/:noteid. The middleware should update the lastsavedtime, title and content of the note in the noteList collection based on the noteid carried in the URL, the current time on the server and the data contained in the body of the request message. Return the lastsavedtime to the client if success and the error if failure.

6. **HTTP GET requests for** http://localhost:3001/searchnotes?searchstr=xx. The middleware should find in the noteList collection all notes of the current user (based on the value of "userId" session variable) whose title or content contains the searchstr carried in the URL. Send _id, lastsavedtime and title of those notes in JSON to the client if database operations are successful, and the error if failure.

7. **HTTP DELETE requests for** http://localhost:3001/deletenote/:noteid. The middleware, should delete the note from the noteList collection according to the noteid carried in the URL. Return an empty string to the client if success and the error if failure.

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 **React_I_examples.pdf**, create a React app named **noteapp** and install the jQuery module in the React app.

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

index.js **(3 marks)**

Modify the generated **Index.js** in the ./src folder of your react app directory, which should render the component you create in **App.js** in the "root" division in the default index.html, and remove any unnecessary code.

App.js **(50 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 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 iNoteApp. In iNoteApp, you may use a state variable to indicate if 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 user has logged in, and update it when a 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. 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. You can limit the number of characters in each note title in the left panel to a small fixed number n, i.e., only the first n characters of the note title is shown and the rest shown as "…".

- 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 a note title in the left panel of a page view as in Figures 2-6, send an HTTP GET request for http://localhost:3001/getnote?noteid=xxx, where xxx should be the _id of the note that you store with the note title in the list. Then render a page view as in Fig. 5.

- When handling the onClick event on the "Save" button in a page view as in Fig. 3, send an HTTP POST request for http://localhost:3001/addnote carrying the new note's title and content in the body of the request message, and handle the response accordingly.

- 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/savenote/xxx where xxx should be the _id of the note being updated, and handle the response accordingly.

- When handling the onClick event on the "Delete" button in a page view as in Fig. 4 or Fig. 5, send an HTTP DELETE request for http://localhost:3001/deletenote/xxx (where xxx is _id of the note to be deleted). If success response is received, update

the page view accordingly.

- When handling the onKeyUp event (event.key == "Enter") on the search input box in a page view as in Figures 2-6, send an HTTP GET request for http://localhost:3001/searchnotes?searchstr=xxx (where xxx is the input search string). When success response is received, update the page view accordingly.

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

NoteService/app.js
NoteService /routes/notes.js
noteapp/src/App.js
noteapp/src/index.js
noteapp/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.