COMP3322A Modern Technologies on World Wide Web

Assignment 1 (22%)

[Learning Outcome 2]
Due by: 23:59, Friday November 11 2022

Overview

In this assignment, you are going to develop a web-based **Album sharing app** using NodeJS/Express.js, JavaScript/jQuery, AJAX, HTML and CSS. The application implements a few simplified functionalities, including displaying photos/videos in albums, switching between different album pages, and posting likes on a photo/video. The main workflow of the **Album app** is as follows.

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

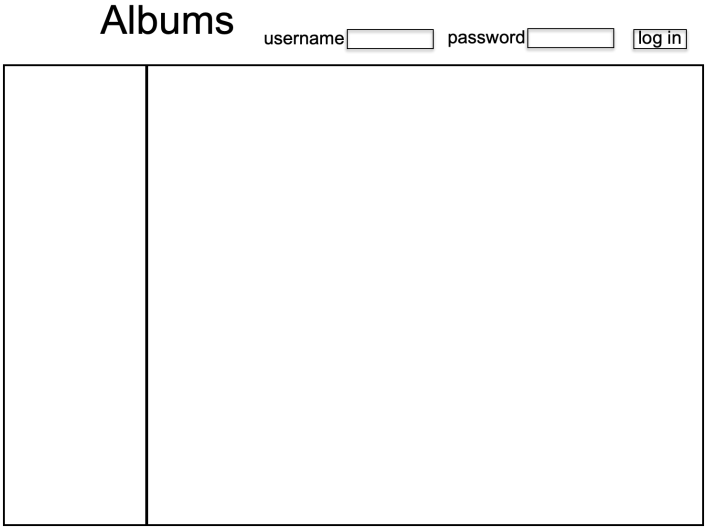


Fig. 1

- After a user has logged in, the sketch of the page is in Fig. 2. A list of friend albums is shown in the left division.

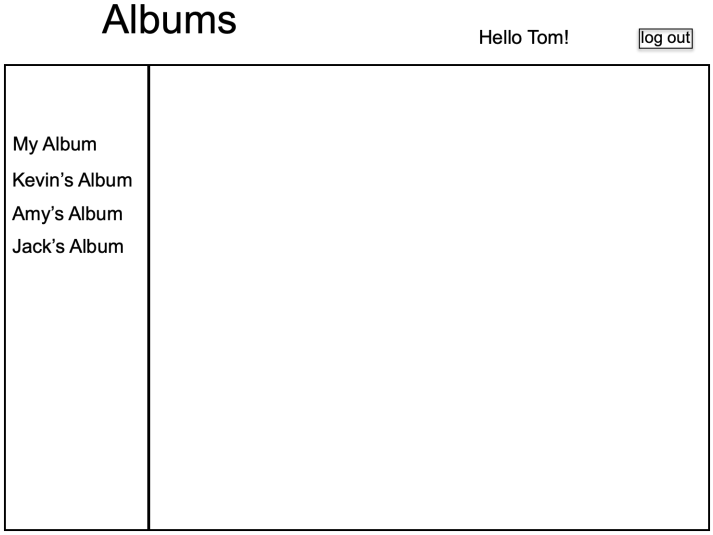


Fig. 2

- After clicking “My Album”, the sketch of the page is in Fig. 3. The photos/videos in the user’s own albums are displayed in the right division, together with messages of who liked a photo/video. If there are more than one page of photos/videos, the user can click “<previous” or “next>” to go to the previous or next page of photos/videos in this album.

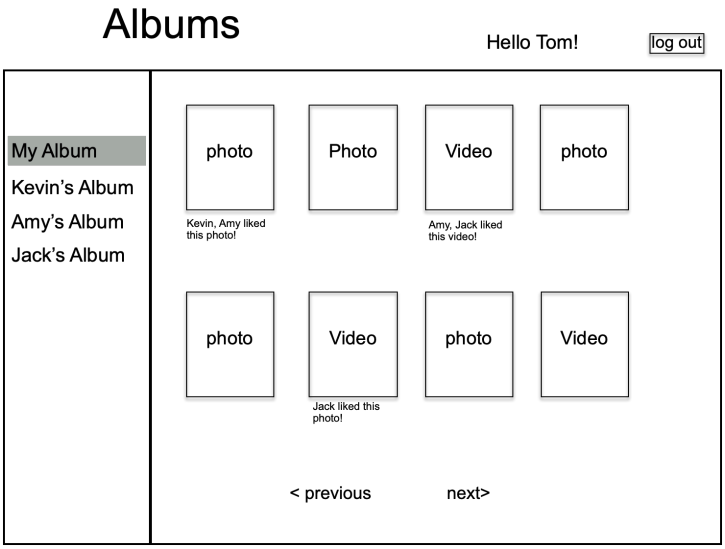
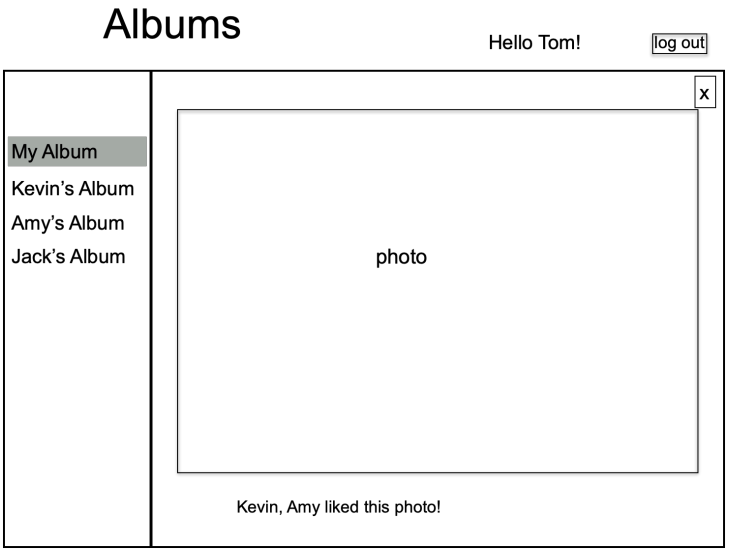


Fig. 3

Each photo/video is clickable. After clicking a photo/video, the photo/video will be enlarged as in Fig. 4, together with the “liked” message. When the cross X is clicked, the page returns to the view as shown in Fig. 3 (the page before the photo/video is clicked).



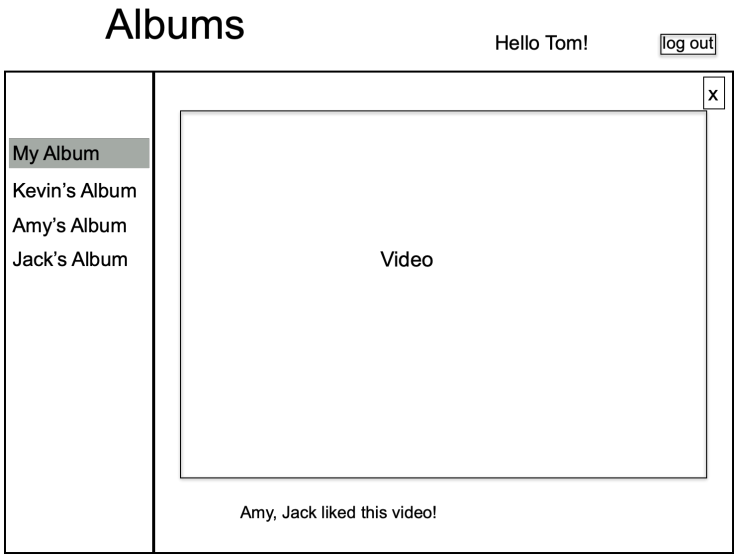


Fig. 4

- When a friend's album is selected in the left-hand list, the page view becomes one in Fig. 5. Photos/videos of the friend are shown in the right division, together with messages of who liked a photo/video and the like buttons. ("<previous" and "next>" are not shown if an album has less than one page of photos/videos, and shown, otherwise.)

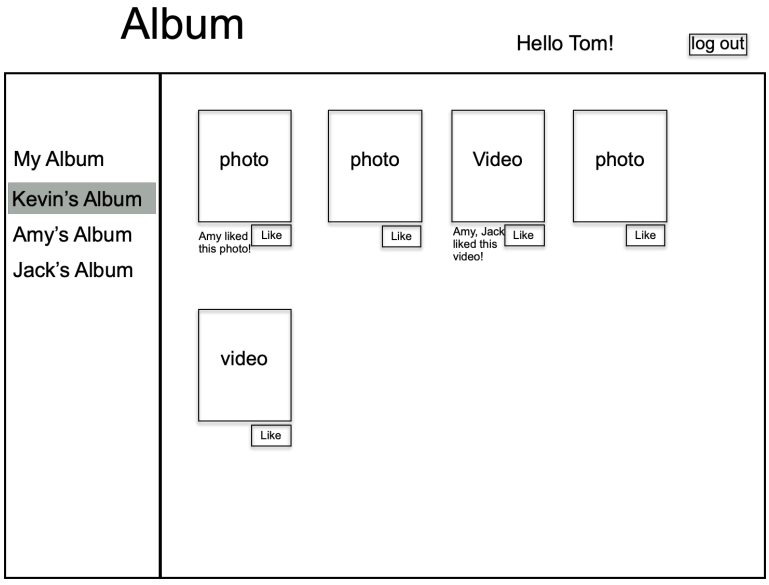


Fig. 5

Similarly, when a photo/video is clicked, the photo/video will be enlarged as in Fig. 6, together with the "liked" message and the like button. When the cross is clicked, the page returns to the view shown in Fig. 5 (the page before the photo/video is clicked).

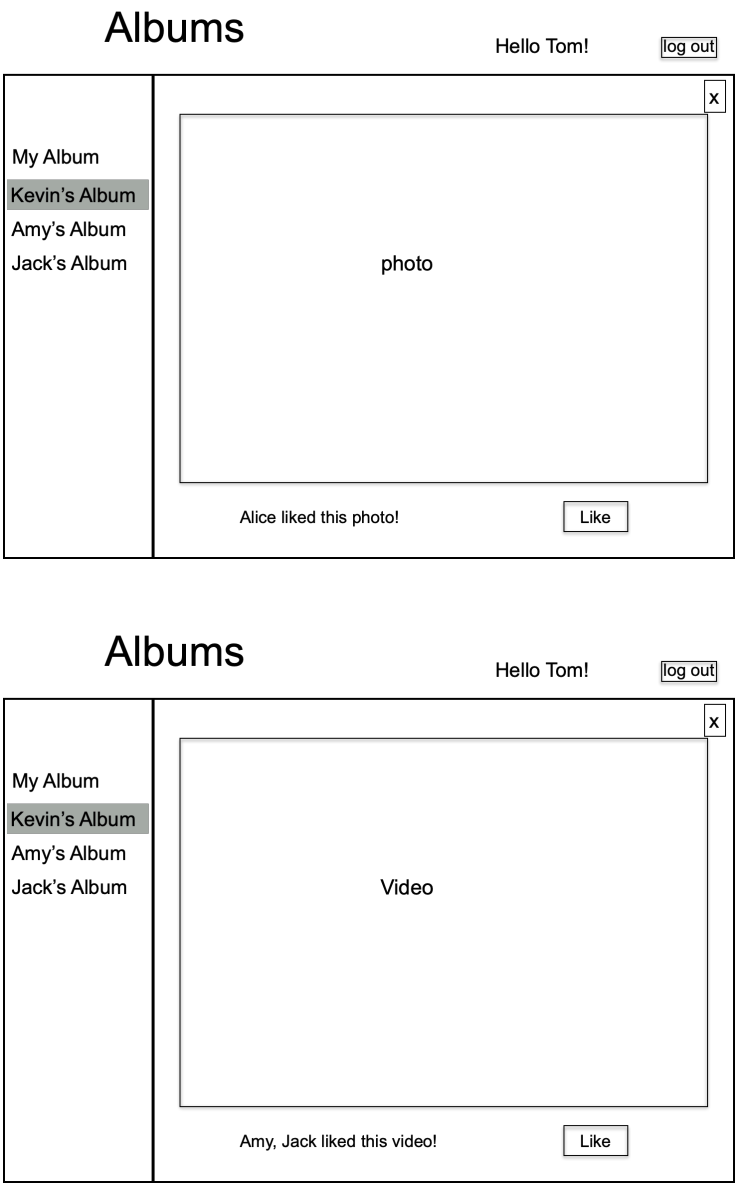


Fig. 6

The web-based album sharing application is to be implemented by the following code in an Express app:

app.js
./public/albums.html
./public/javascripts/script.js
./public/stylesheets/style.css

and accessed at http://localhost:8081/albums.html.

Preparations

1. Following steps in **setup_nodejs_runtime_and_examples_1.pdf**, install the Node.js environment and the Express framework, and create an Express project named **AlbumApp**.

2. Following steps in **AJAX_JSON_MongoDB_setup_and_examples.pdf**, install MongoDB, run MongoDB server, and create a database "assignment1" in the database server.

3. Insert a number of 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({'username': 'Tom', 'password': '123456', 'friends':['Kevin', 'Amy', 'Jack']})
```

Create a folder "media" under the public folder in your project directory. Copy a number of photos and videos to the media folder (due to our simplified implementation to be outlined later, we will only use .jpg photo files and small-size mp4 video files in this assignment). Insert a number of records to a mediaList collection in the database in the format as follows, each corresponding to one photo/video you have copied to the media folder. Here userid should be the value of _id generated by MongoDB for the photo owner's record in the userList collection, which you can find out using db.userList.find().

```
db.mediaList.insert({'url': 'http://localhost:8081/media/1.jpg', 'userid': 'xxxxxxx', 'likedby':['Kevin', 'Amy']})
db.mediaList.insert({'url': 'http://localhost:8081/media/2.mp4, 'userid': 'xxxxxxx', 'likedby':[]})
```

Insert enough records for your testing of the app functionality. For implementation simplicity, in this assignment, we do not store photos/videos in MongoDB. Instead, we store them in the harddisk under the ./public/media/ folder, and store the path of a photo/video in the mediaList collection only, using which we can retrieve the photo/video in the media folder.

Task 1 (35 marks) Implement server-side logic (app.js)

In **app.js**, 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.

Add the middleware to serve requests for static files in the public folder.

We will let the server run on the port 8081 and launch the Express app using command "node app.js".

Implement the following middlewares to handle different client requests:

1. **HTTP GET requests for** http://localhost:8081/load. The middleware checks if the "user_id" cookie has been set for the client. If not, send an empty string back to the client. Otherwise, retrieve username of the current user (according to the value of the "user_id" cookie), and username and _id of friends of the current user from the userList collection in the database; send all retrieved information as a JSON string to the client if the database operation is

successful, and the error message if failure. You should decide the format of the JSON string and extract data from it accordingly in the client-side code to be implemented in Task 2.

2. **HTTP POST requests for** http://localhost:8081/login. 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, set a cookie with the key of "user_id", the value of this user record's _id, and a maxAge of 30 minutes. Send the username and _id of all friends of this logged-in user as a JSON string in the body of the response message if database operation is successful and the error message if failure. Again, you should design the format of the JSON string.

3. **HTTP GET requests for** http://localhost:8081/logout. The middleware should unset the "user_id" cookie, and send an empty string back to the user.

4. **HTTP GET requests for** http://localhost:8081/getalbum?userid=xx&pagenum=xx. If the userid in the URL is "0", the middleware should retrieve _id, url and likedby array of selected photos/videos of the current user from the mediaList collection in the database, based on the userid stored in the "user_id" cookie and the pagenum in the URL. Otherwise, retrieve _id, url and likedby array of selected photos/videos of the respective friend from the mediaList collection based on the userid and pagenum in the URL. You can decide the number of photos/videos per page to show in your web page (e.g., 2X4 as in Fig. 3), and then the total number of pages in each album is decided by the total number of photos/videos you store for each album in mediaList divided by the number of photos/videos to show per page. Retrieve information of the corresponding page of photos/videos from the respective album from the mediaList collection according to pagenum in the URL and your setting of the number of photos/videos per page. Assume the pagenum starts from 0. Send all retrieved information, together with the total number of pages of the respective album, as a JSON string in the body of the response message if database operation is successful, and the error message if failure. You should decide the format of the JSON string to be included in the response body.

5. **HTTP POST requests for** http://localhost:8081/postlike. The middleware should update the likedby array of the photo/video record in the mediaList collection, corresponding to the photovideoid in the body of the POST request message. Especially, use _id of the current user in the "user_id" cookie to find out the username of the current user from the userList collection, and then use the db.collection.update method (see more usage of the update method at https://automattic.github.io/monk/docs/GETTING_STARTED.html) to update the respective photo/video record in the mediaList collection, if the username has not been contained in its likedby array. Return the updated likedby array to the client if success and the error message if failure.

Task 2 (55 marks) Implement client-side logic (./public/albums.html, ./public/javascripts/script.js)

Create **albums.html** which renders a layout as sketched in Fig. 1, which contains the album heading, a username text input box, a password input box, a "log in" button and two divisions. **albums.html** should link to the JavaScript file **javascripts/script.js** and the stylesheet **stylesheets/style.css**. Register an event handler "init()" with the "onload" event of the <body> element in **albums.html**.

In **script.js**, implement the following functionalities, such that the client-side page views and functionalities as illustrated in Figures 1-6 can be achieved.

1. **Page load.** Implement the init() function which sends an AJAX HTTP GET request for http://localhost:8081/load, and then: (1) if an empty string is received from the server, a page as shown in Fig. 1 should be displayed; (2) if a JSON string carrying username of the logged-in user and _id and username of the friends is received, a page as shown in Fig. 2 should be displayed (the same page view as after successful login in Point 2 below). Especially, the username of the logged-in user is to be shown in the message "Hello username!" in the header area. You can decide the HTML elements to render, in order to implement the respective page view.

2. **Log in**. Register an event handler "login()" with the "onclick" event on the "log in" button on the page view as shown Fig. 1. Implement the function login() as follows: Retrieve the input username and password in the respective input boxes. If either username or password input is empty, an alert box pops up, displaying "Please enter username and password", and the function exits. Otherwise, send an AJAX HTTP POST request for http://localhost:8081/login, carrying the input username and password. If the string returned by the server side is "Login failure", pop up an alert box which shows "Login failure" and the page view remains the one in Fig. 1; otherwise, replace username and password input boxes by "Hello xx!" (xx is the username of the logged-in user) and a "log out" button, and display the friend album list according to information received from the server side (Fig. 2). Further, use the localStorage object to store the page number that the user views on each album: you can store (username of the album owner, current page number) into the localStorage object (see https://www.w3schools.com/jsref/met_storage_setitem.asp); initialize the current page numbers to 0.

3. Log out. Register an event handler "logout()" with the "onclick" event on the "log out" button. Implement the function logout() as follows: Send an AJAX HTTP GET request for http://localhost:8081/logout. When an empty string is received from the server, display the page view as in Fig. 1. Clear the data you have stored in localStorage.

4. **Display album**. Register an event handler "loadAlbum(event, 0)" with the "onclick" event on each album in the album list. Implement the function loadAlbum(event, page) as follows: If it is "My Album" that is clicked, send an AJAX HTTP GET request for http://localhost:8081/getalbum?userid=0&pagenum=page; if it is a friend's album that is

clicked, send an AJAX HTTP GET request for http://localhost:8081/getalbum?userid=id&pagenum=page, where id is the _id of the friend in the userList collection in the database (**Hint**: you may store _id and username of each friend as attributes in each album entry element, such that you can retrieve them through event.target.xx, where xx is the attribute in which you store the _id or username). When the JSON string containing _id, url and likedby array of the respective album page's photos/videos is received, display the photos/videos following the sketch in Fig. 3 or Fig. 5. Especially, each photo is an <img> element with a "src" attribute pointing to the url of the respective photo, and an "id" attribute recording _id of the photo; each video is a <video> element with a "src" attribute pointing to the url of the respective video, an "id" attribute recording _id of the video, and the "controls" attribute. If the likedby array of a photo/video is not empty, display a message "xx liked the photo (or video)!" underneath the photo/video; otherwise, do not display the message. You can decide the display size of the photos/videos, the number of photos/videos to display on each page (to be consistent with the setting on the server side), and the order of photos/videos (e.g., alphabetic order of the file names). Note that the album entry in the list in the left division, whose photos/videos are being displayed in the right division, should be highlighted. The "<previous" link and the "next>" link should be displayed according to the current page number and the total number of pages in the respective album as follows: (1) if the current page is 0, the "<previous" link should be greyed out and not clickable; (2) if the current page number equals the total number of pages minus 1, the "next>" link should be greyed out and not clickable; (3) if the current page is 0 and the total number of pages equals 1, the "<previous" link and the "next>" link should not be displayed. In addition, you should set page number of the respective album stored in localStorage to the page value in loadAlbum(event, page).

5. **Previous Next**. Register an event handler "loadAlbum(event, localStorage.username-1)" with the "onclick" event on the "<previous" link, and register an event handler "loadAlbum(event, localStorage.username+1)" with the "onclick" event on the "next>" link, where username should be the username of the respective album owner. Clicking the "<previous" link or the "next>" link will invoke the same function loadAlbum(event, page) as implemented in Point 4 above.

6. **Display individual photo/video**. Register an event handler "displayPhoto(event)" with the "onclick" event on each photo on the page view in Fig. 3 or Fig. 5. Register an event handler "displayVideo(event)" with the "onclick" event on each video on the page view in Fig. 3 or Fig. 5. In each event handler function, change the display in the right division according to the page view in Fig. 4 or Fig. 6, displaying the photo or video together with the "liked" message (if any), the "Like" button (only in cases of viewing friend's album as in Fig. 6), and the "x" in the right division. The video should be displayed with controls. You can add more arguments to displayPhoto(event) and displayVideo(event) functions to pass in needed information in implementing the functions.

Register an event handler loadAlbum(event, localStorage.username) with the "onclick" event on the "x", such that the page view goes back to the page of photos/videos before the user

clicks an individual photo/video for display, when "x" is clicked.

7. **Post like**. Register an event handler "handleLike(event)" with the "onclick" event on each "Like" button (on the page view in Fig. 5 or 6). Implement the handleLike(event) function as follows: Send an AJAX HTTP POST request for http://localhost:8081/postlike, carrying photovideoid=xx in the body of the request message, where xx should be _id of the photo/video in the mediaList collection in the database. Upon receiving the updated likedby array in the response, display the updated "liked" message underneath the photo/video on the respective page.

Task 3 (10 marks) Style the page using CSS

Please use a separate **style.css** file to include all your styling rules.

1. (5 marks) Use CSS styling you choose to make your page look nice with good layout;

2. (5 marks) Implement Responsive Web Design to make your page look nice on screens of different sizes.

Notes:

1. You can use either basic JavaScript or jQuery to implement client-side scripting.

2. Maintain a good programming style: avoid redundant code; the code should be easy to understand and maintain.

3. You should carefully test all the functionalities stated in this handout.

Submission

You should submit the following files in the specified folder structure in a zip file:

(1) app.js

(2) /public/albums.html

(3) /public/javascripts/script.js

(4) /public/stylesheets/style.css

Please submit the .zip file on Moodle:

(1) Login Moodle.

(2) Find "Assignments" in the left column and click "Assignment 1".

(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.