COMP3322A Modern Technologies on World Wide Web

Assignment 1: A Simple Webmail System (20%)

[Learning Outcomes 1, 2]

Due by: 23:59, Wednesday November 6, 2019

Total mark is 100.

Overview

In this assignment, you are going to develop a simple **web-based email system** using NodeJS/Express.js, MongoDB, JavaScript, AJAX, HTML and CSS. The webmail system implements a few simplified functionalities, including displaying emails in your mailboxes, moving emails from one mailbox to another and compose new messages. The web-based email system is to be implemented by the following code in an Express app:

app.js

./public/webmail.html

./public/javascripts/script.js

./public/stylesheets/style.css

and accessed at http://127.0.0.1:8081/webmail.html.

Preparations

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

2. Following steps in Lab 4, install MongoDB, run MongoDB server, and create a database "assignment1" in the database server.

Insert a number of email documents into an emailList collection in the database in the format as follows:

```
db.emailList.insert({'sender':'amy@cs.hku.hk', 'recipient':'john@cs.hku.hk',
'title':'Meeting today', 'time':'20:32:01 Fri Oct 11 2019', 'content':'Shall we meet at the
library at 5pm?', 'mailbox':'Inbox'})
```

Each email document in the emailList collection contains the following key-value pairs:

- **_id**: The unique ID of the document, which you do not need to include when inserting a document, and MongoDB server will add automatically into each inserted document. You can find out such a _id using db.emailList.find().

- **sender** – The email address of the sender.
- **recipient** – The email address of the recipient.
- **title** – The title of the email.
- **time** – The time when the email was sent.
- **content** – The content of the email.
- **mailbox** – The mailbox which the email belongs to, among Inbox, Important, Sent and Trash.

Please prepare at least 3 pages of emails in each mailbox in the emailList collection. That is to say, if you set the number of emails to display on each page in the email division to be 3, then you should prepare at least 9 email documents for each mailbox in the collection.

Task 1 (15 marks) Create basic webmail.html and app.js

1. *Client side*

Create **webmail.html** which renders the layout shown in the following figure, i.e., the page has four main divisions: heading, function buttons, mailbox list and email list. You can design any heading, showing the name of your webmail system. In the function division, there should be four buttons: "Compose", "Move to …", "<" and ">". You can implement the buttons using any elements you like, as long as they are clickable and upon clicking, the corresponding functionality can be achieved. In the mailbox division, include four links: "Inbox", "Important", "Sent" "Trash". In the email division, the email entries will be dynamically loaded from the server side, to be implemented in Task 2.

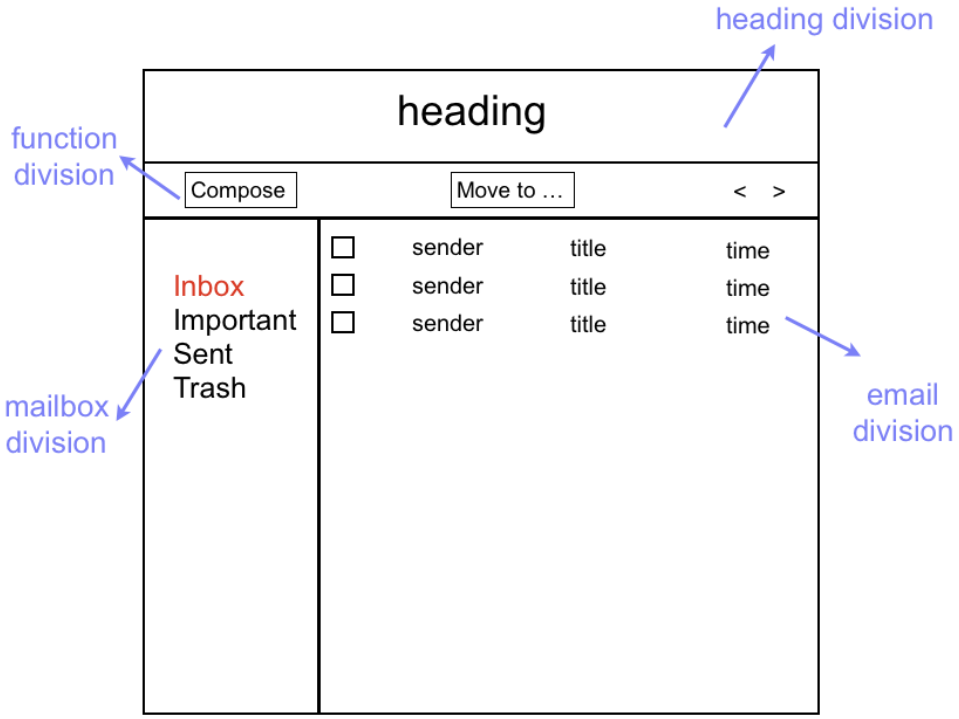

Fig. 1

2. *Server side*

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 for serving static file in the "./public" directory.

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

Task 2 (25 marks) Display the email list

When **webmail.html** is first loaded, it should be showing the "Inbox" view (Fig. 1): The "Inbox" link in the mailbox division should be in a different color (e.g., "color 1") from the color of the other mailbox links (e.g., "color 2"); when the "Move to …" button is clicked, a drop-down list should be shown, containing three entries: "Important", "Sent", and "Trash". A fixed number of emails in the inbox are listed in the email division, following the format shown in Fig. 1 (checkbox, sender, title, time). You can specify the number of emails for display on a page.

After you click "Important" in the mailbox division, the web page should be showing the "Important" mailbox view: The "Important" link turns into color 1 and the other three links are in color 2; when the "Move to …" button is clicked, the drop-down list contains three entries: "Inbox", "Sent", and "Trash". A fixed number of emails in the "Important" mailbox will be listed in the email division according to the format in Fig. 1 and the number you specified.

When you click "Trash" in the mailbox division, the web page should be showing the "Trash" mailbox view: The "Trash" link turns into color 1 and the other three links are in color 2; when the "Move to …" button is clicked, the drop-down list contains three entries: "Inbox", "Sent", and "Important". A fixed number of emails in the "Trash" mailbox will be listed in the email division according to the format in Fig. 1 and the number you specified.

When you click "Sent" in the mailbox division, the web page should be showing the "Sent" mailbox view (Fig. 2): The "Sent" link turns into color 1 and the other three links are in color 2; when the "Move to …" button is clicked, the drop-down list contains three entries: "Inbox", "Important" and "Trash". A fixed number of emails in the "Sent" mailbox will be listed in the email division according to the format in Fig. 2 (checkbox, recipient, title, time) and the number you specified.

If there are more than one page of emails in the corresponding mailbox, emails on other pages are displayed when you click "<" or ">" in the function division.

Please note that switching the mailbox view, and clicking "<" or ">" should load the

corresponding email list in the email division **without reloading the entire web page.**

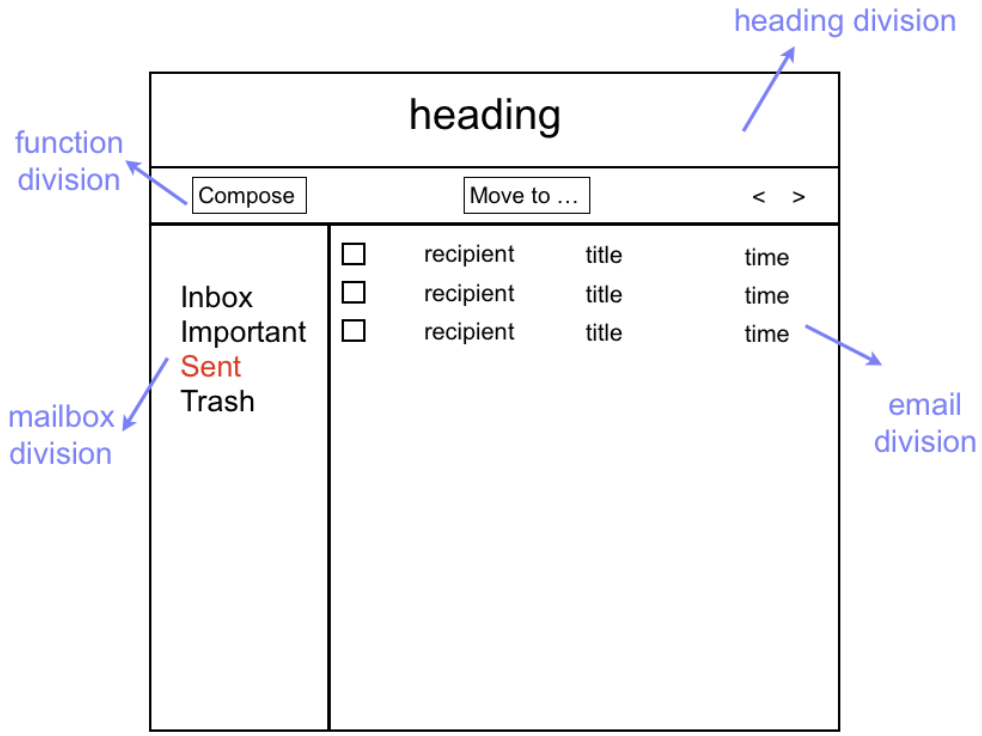

Fig. 2

Implement the following to achieve the above functionalities:

1. *Client Side.*

1.1 When **webmail.html** is first loaded, load your specified number of emails in the Inbox in the email division, and order them in reverse chronological order. (**Hint**: register a JS event **onload** in the <**body**> tag)

1.2 When a mailbox link in the mailbox division is clicked, retrieve your specified number of the newest emails in the corresponding mailbox from the server side and display them in the email division, order them in reverse chronological order, and display the corresponding function buttons in the function division. (**Hint**: use the JS event **onclick**).

1.3 Whenever "<" or ">" is clicked, newer or older emails in the corresponding mailbox (of your specified number) will be retrieved and listed in the email division. (**Hint**: use the JS event **onclick**).

1.4 Implement AJAX codes in the event handler functions to communicate with the server for retrieval of emails from the database.

2. *Server Side.*

In **app.js**, create the following middleware to handle requests from the client side:

- **HTTP GET requests for** http://127.0.0.1:8081/retrieveemaillist. In the middleware,

retrieve information of a number of emails in the respective mailbox from the emailList collection in the database, according to the request from the client, and send them to the client (note that only necessary information for the email list display should be retrieved and sent to the client, e.g., the email content is not retrieved/sent). You should decide what data the HTTP request should carry for the server side to retrieve emails from the database accordingly, and the format of the data sent back to the client in the response body.

Task 3 (15 marks) Display email content

When you click each email entry in an email list display (on sender/recipient, title or time), a request will be sent to the server, which will retrieve the content of the email from the emailList collection in the database, and send it back to the client. The client should display the content in the email division, replacing the email list, in the format as shown in Fig. 3. This should not cause reload of the entire web page as well.

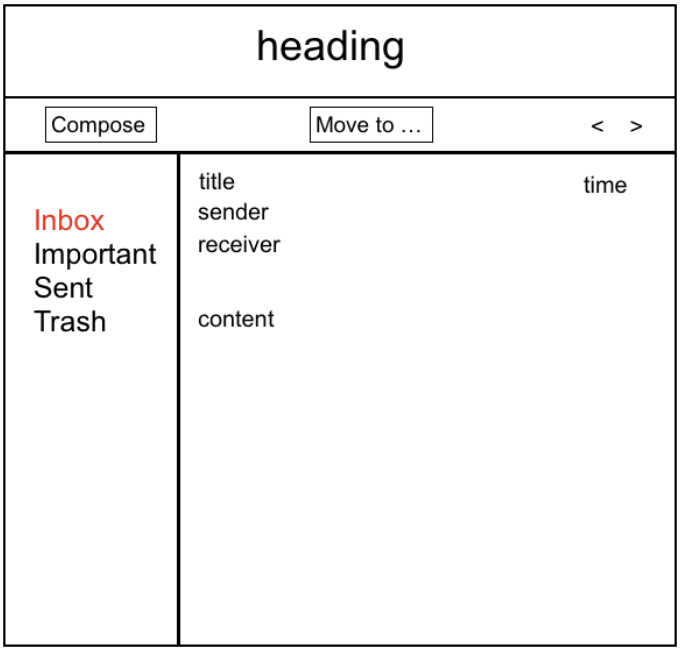

Fig. 3

On this page view, when "<" or ">" is clicked, the content of the previous (i.e., immediately newer) email or the next (i.e., immediately older) email in the same mailbox will be retrieved and displayed. If the current email is the newest one when "<" is clicked or the oldest one when ">" is clicked, the page view remains unchanged.

1. *Client Side.*

Similar to Task 2, you should decide the JS events to trigger, implement the corresponding event handling AJAX code on the client side.

2. *Server Side.*

In **app.js**, create the following middleware to handle corresponding requests from the client side:

- **HTTP GET requests for** http://127.0.0.1:8081/getemail. In the middleware, retrieve information of the specific email from the emailList collection in the database, according to the request from the client, and send them to the client. You should decide what data the HTTP request should carry for the server side to retrieve the correct email from the database accordingly, and the format of the data sent back to the client in the response body.

Task 4 (20 marks) Move emails

To the left of each email entry in an email list display, there is a checkbox, which you can click to check and uncheck. When the web page is displaying a particular mailbox view, after you select some emails in the email list and select an entry in the drop-down list of "Move to …", a request will be sent to the server to update the "mailbox" field of the corresponding email documents to the selected mailbox correspondingly.

After these updates have been successfully done on the server side, the server will further retrieve information of Y older emails (if you moved Y emails in the current email list display), and send them to the client. On the client side, you should remove the selected email entries from the email list display in the email division, and append the Y newly retrieved email entries to the bottom of the list, such that the total number of email entries displayed is still the same as before (or smaller only if all older emails have been retrieved). All these should not cause reload of the entire web page as well.

On an email content page view as shown in Fig. 3, if you select an entry in the drop-down list of "Move to …", the current email being displayed should be moved to the selected mailbox, i.e., a request should be sent to the server to update the "mailbox" field of the corresponding email document to the selected mailbox. Then the page view should go back to the email list display as shown in Fig. 1 or Fig. 2, showing the first page of emails in the mailbox which the moved email originally belonged to.

1. *Client Side.*

Similar to previous tasks, you should decide the JS event to trigger, and implement the event handling AJAX code on the client side.

2. *Server Side.*

In **app.js**, create the following middleware to handle corresponding requests from the client side:

- **HTTP POST requests for** http://127.0.0.1:8081/changemailbox. In the middleware, update the mailbox field of the corresponding email document(s) in the emailList

collection, according to the request from the client; then retrieve information of emails according to the email list to be generated on the client side, and send them to the client. You should decide what data the request should carry for the server side to update the email document(s) accordingly, and the format of the data sent back to the client in the response body.

Task 5 (15 marks) Compose new email

When the "Compose" button is clicked, the page view should be as shown in Fig. 4. The "Move to …", "<" and ">" buttons should be disabled on this page view. You can compose a new email message on the page; when "Send" is clicked, a request should be sent to the server side for inserting a new email document into the emailList collection, containing information of the newly composed email. Upon receiving success response from the server side, the page view should go back to the email list display as shown in Fig. 1 or Fig. 2, showing the first page of emails in the mailbox – the one whose email or email list you were viewing before clicking the "Compose" button.

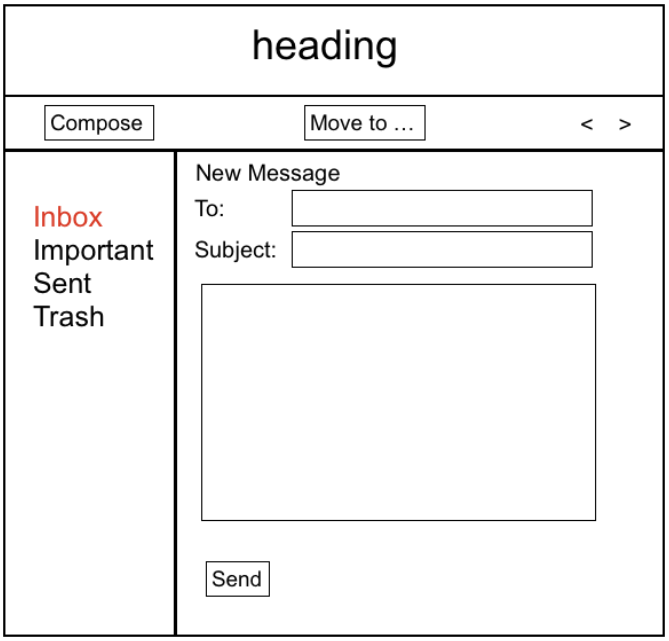

Fig. 4

1. *Client Side.*

Implement the "To" and "Subject" fields using text input boxes and the message content field using <textarea>. When the "Send" button is clicked, an HTTP post request should be sent to the server side. Similar to previous tasks, you should decide the JS event to trigger, and implement the AJAX code on the client side.

2. *Server Side.*

In **app.js**, create the following middleware to handle corresponding requests from the client side:

- **HTTP POST requests for** http://127.0.0.1:8081/sendemail. In the middleware, insert a new email document into the emailList collection in the database, according to the information of the newly composed email carried in the request body. Besides, you can use the same default sender email address for all composed emails; the "time" in the new email document should be the current time on the server side; the "mailbox" in the new email document should be "Sent". Then retrieve information of emails according to the email list to be generated on the client side, and send them to the client. You should decide what data the HTTP request should carry for the server side to insert the new email document accordingly, and the format of the data sent back to the client in the response body.

Task 6 (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/webmail.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.