g4minh Nguyen Quang Minh 1000468059

☺ Design Note ☺

My server will run on http://127.0.0.1:3000/
My application is in a **single** index.html page.
Multiple users can log in and use the app at the same time.
The data is sync between the client and server after each button click.
Please **turn on shared location** feature of your web browser before testing my code.

General flow of my app:

- On the client side, after a client logging into the app, either by signing up for a new account or logging in using their registered email and password, the server will pass two objects to the client:
    - currentUser is the object represents for the client who is using the app.
    - allUsers is an array of all client objects in the database. This array **doesn't contain the password** of any clients. So the current client doesn't have access to the password of other clients in the server.
    - allUsers, currentUser is cleared after the user logout.
- On the server side there are two tables that used to stored the user information:
    - Users is the table that stores the user information (e.g. email, password, display name…). Any clients can make a GET request to get this table, and the server will return this table with the **password section removed**.
    - adminDataOnly is a the table that stores user tracking information (e.g. location, log-in time, profile view count …). This special table is only accessible to the client who status is "admin" or "superadmin". An admin client can make a POST request to retrieve those data using both their email and password. The server will **verify if that email and password belong to an admin** then return a user tracking information. **Why do I use POST instead of GET in this case**? Because POST is much safer when transfer sensitive data such as password.
- Each time the client changes their data, or someone data, a request will be sent to update data stored in table Users in the server. Beside from the information about what to change, **a request to update or delete data in the server also needs to include the requester email and requester password**. This helps the server to identify if the requestor has the permission to change, delete the data that they want. (e.g. basic user cannot make a request to modify admin user data).

- My application runs on a single page, there is no new page or refreshing. So whenever the client clicks a button on a page to do a task, it is counts as "refreshing", a request will immediately be sent to the server to get the new data, which is used to update currentUser and allUsers. * Depending on the state of currentUser and allUsers **after** being updated, the app will check if it still possible to do the wanted task. If it is, then the app will perform the task, otherwise the client will be directed to appropriate page. After the task is completed, if necessary, currentUser and allUsers will be updated once again.

Example test case for *:
User admin A deletes user B that just has been deleted by user admin C when A, B and C are all logged in.

- B then should get a warning and logged out of the app after clicking a button.
- A cannot delete user B, because B no longer exists in the server, so A is redirected to the welcome user page where there is an updated list of users in database with B removed.

A basic user is assigned an admin status when they are in the app. Then that user will immediately has the right of an admin for after clicking some buttons.
API summary:

Method	URL	**Task**
POST	/users/login	Login the user to the app. Can use GET, but POST is safer when dealing with password.
POST	/users/new-account	Create new account for a client.
GET	/users	Getting all user in table Users with their password removed.
GET	/users?email=	Getting a specific user in table Users with password section removed.
PUT	/users/avatars	Update a user avatar
GET	/users/avatars	Get a user avatar using their email.
PUT	/users/profiles	Update a user profile.
PUT	/users/passwords	Update user password.
POST	/users/admins	Give an admin privilege for a user.
DELETE	/users/admins	Remove admin privilege of a user.
DELETE	/users	Delete a user from the server.
PUT	/users/admins/data/viewcounts	Increments the view-count of a user in the table adminDataOnly.
POST	/users/admins/data	This involve passing admin password to retrieve tracking information of the users, thus need to use POST for safety.

Note on user avatar:

- The default avatar is stored in the folder User-avatar.
- The user can upload their new avatar in edit user profile.
- The image uploaded to the server will be saved in User-avatar folder. And the path to the new avatar will be saved to the database.