Drew Carlson

Image Web Hosting

Sometimes, games or websites may seek to have image assets hosted online. This webapp allows users to upload their own images to a database, so that they can be used in any number of places that want images to be webhosted. This app seeks to offer different ways to manage hosted images, allowing users to easily upload and then grab the link as to where the image is hosted, or delete old images.

Ideally, this app would use paid versions of mongoDB and redis to allow for mass image stores, but this demo is somewhat limited in how many images can be stored.

This website would make profit (mainly to fund mongo/redis) by running advertisements, as well as offering a premium subscription. Premium users get increased storage space, and do not get shown ads.

Current Implementation:

This app was made by extending DomoMaker E. It is now fairly expanded, but still uses the styling and images from domomaker. The only references to 'domo' in the code currently are class names and id's, so the styling remains.

After making an account, users can upload images they would like to have hosted online. These images are displayed in a list to the user and clicking on a list item brings up the image. Users can change password and can upgrade to premium to get additional file uploads. Currently, ads are still displayed for premium users, and images cannot be deleted.

React is being used to dynamically load the login, signup, upload image, change password, and upgrade account forms, as well as to load in advertisements and a list of images a user has uploaded. There is also a 'blank' component, but I imagine react likely has something built-in for this.

MongoDB currently stores user accounts and images user's upload. GridFS is currently not being used.

Handlebars is currently being used to load in the image list dynamically, as well as place csrf tokens.

The next big step is to redo the styling and base structure of the app page, allowing for nice looking advertisements and eliminating all mentions of domo in the code. Many post methods need more meaningful feedback for success. I plan to expand my usage of handlebars to display the user's remaining image uploads, load in an image preview for every uploaded image, and stop ads for premium users. Users need to be able to remove uploaded images to free up space, and the presentation of uploaded images should be improved to allow for easy management (and likely provide a copyable link for easy access). I may move image uploads to its own page. If I have enough time, I would like to implement GridFS for above and beyond credit.

KNOWN BUGS:

- Signing up is currently bugged and doesn't set the session user properly, crashing the app. However, you are still successfully signed up after this, so just go to the login page and log in. This issue is not present in login or changing passwords.
- MongoDB errors are not handled properly and crash the app.

ENDPOINTS:

- /getToken
  - Supported Methods: GET
  - Query Params: N/A
  - Description: Retrieves a CSRF token for secure logins
  - Supported Return Types: JSON
- /login
  - Supported Methods: GET, POST
  - Query Params: N/A
  - Body Params: user (username), pass (password)
  - Description: Loads the login page; posts login info to access the user's account
  - Supported Return Types: N/A; JSON
- /signup
  - Supported Methods: POST
  - Body Params: user (username), pass (password), pass2 (confirm password)
  - Description: sends the info to create an account to mongDB
  - Supported Return Types: JSON
- /logout
  - Supported Methods: GET
  - Query Params: N/A
  - Description: Logs the current user out
  - Supported Return Types: N/A
- /uploads
  - Supported Methods: GET
  - Query Params: N/A
  - Description: Loads the application's main page where users can upload images and see the list of uploaded images
  - Supported Return Types: N/A
- /getImages
  - Supported Methods: GET
  - Query Params: N/A
  - Description: Retrieves API data for each image a user has uploaded (i.e. name, mimetype)
  - Supported Return Types: JSON
- /image
  - Supported Methods: GET
  - Query Params: image (name of the image file)
  - Description: retrieves an image from the database by name
  - Supported Return Types: image/* (based on the image's mimetype)
- /upload
  - Supported Methods: POST
  - Body Params: pic (uploaded image file)
  - Description: uploads a picture, which is stored in the database

    - Supported Return Types: JSON
- /upgrade
    - Supported Methods: POST
    - Body Params: N/A
    - Description: Upgrades the signed in user's account to premium
    - Supported Return Types: JSON
- /updatePass
    - Supported Methods: POST
    - Body Params: pass (existing password), newPass (new password), newPass2 (confirm new password)
    - Description: changes the password for the signed in user
    - Supported Return Types: JSON