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.

**Final Implementation:**

This app was made by extending DomoMaker E. It is a web service where users can host and manage images.

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.

React is being used to dynamically load the login, signup, upload image, change password, and upgrade account forms, remaining uploads, as well as to load in advertisements and a list of images a user has uploaded. Ads are not displayed to premium users; they get a thank you message instead.

MongoDB stores user accounts and images user's upload.

Handlebars is being used to support react, only loading in the current window (app/login).

I ended up not having the time to implement GridFS. As for above and beyond, the thing I can think of best applying was that I put hosted image URLs in text boxes and gave users the ability to copy them to clipboard, using a very helpful w3Schools tutorial. The image displays were somewhat annoying to get working, although that's base application stuff.

**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
- /user
    - Supported Methods: GET
    - Query Params: N/A
    - Description: Returns non-sensitive information about the current user (i.e. account type, remaining slots)
    - Supported return types: JSON
- /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
- /remove
    - Supported Methods: POST
    - Body params: _id: the mongoDB id of an image to remove.
    - Description: Deletes the specified image from the database and returns an image slot to the user
    - Supported Return Types: JSON
- *
    - Supported Methods: GET
    - Query Params: N/A
    - Description: Handles 404 errors for non-existent pages, redirecting users to login/ uploads
    - Supported return types: N/A