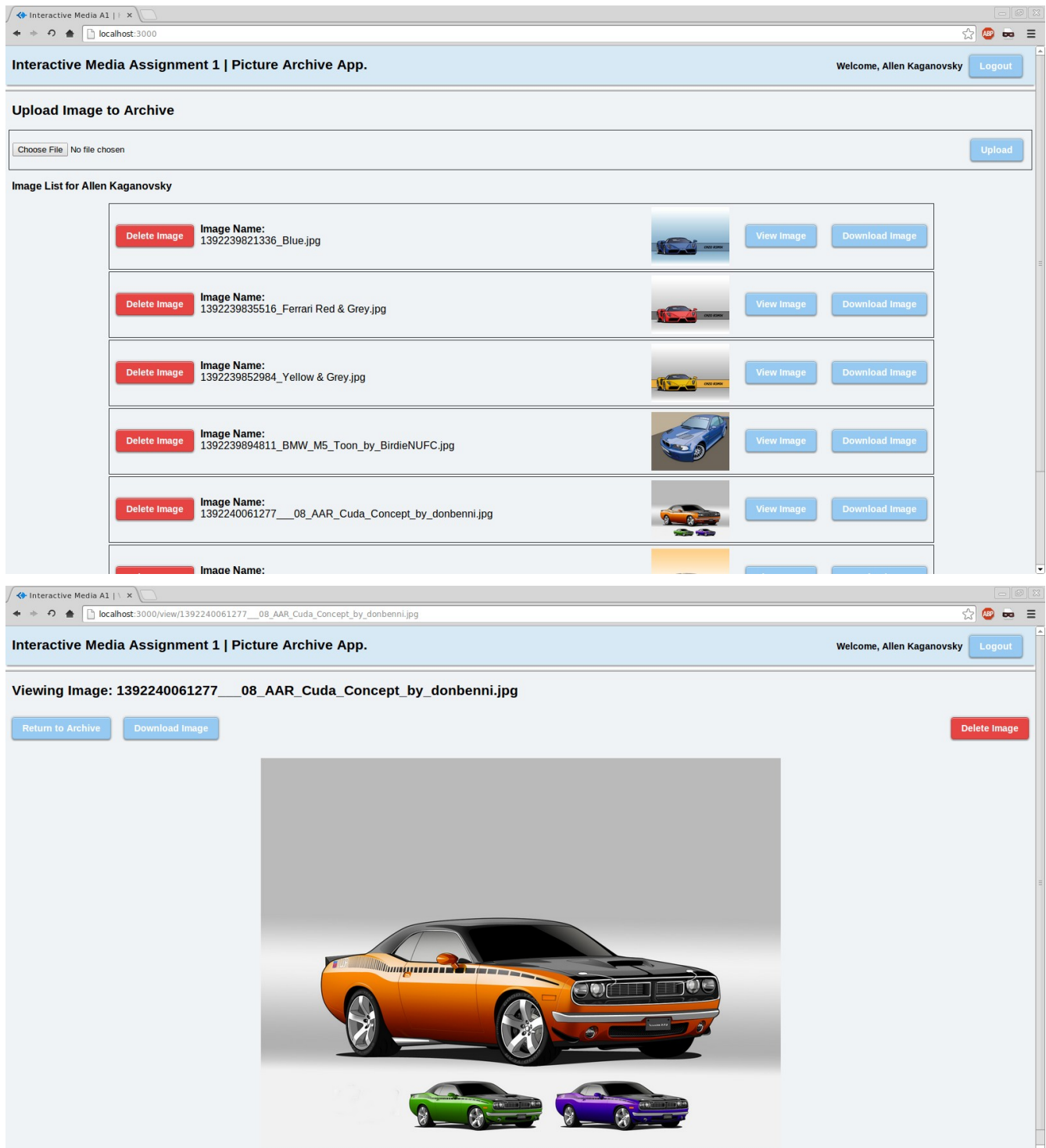


# Interactive Media Assignment #1

By: Allen Kaganovsky [ 100389429 ]

## Assignment #1 : Picture Archive Application using NodeJS

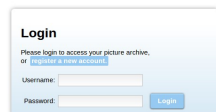
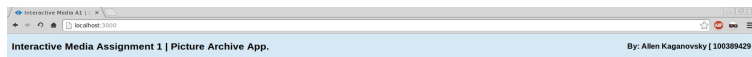
Live demo is available at: <http://www.kaganovsky.ca:3000>



My application written in NodeJS as the backend uses two npm modules, the express module as well as the ejs template module, furthermore, the 'fs' built in module is also used extensively for image storage and retrieval for each user as well as persistent storage of registered users by overwriting the users.json file. In order to examine my application more in depth, we can divide the application into several sections; the user authentication and registration sections, the image upload and delete methods, and lastly, the users main home page where they are able to view all of their archived images.

## User authentication and registration

When a user first accesses the application, the application checks if a session name identifying the user has been set. If it has already been set, the user is presented with the home page displaying their personal archived images. If the user has not yet logged in then they are presented with the login page, the user also has the option to register a new account for themselves. When registering a new account, the register ejs template posts the username and password to the application backend, if the username is already found within the users.json file – an error is displayed and the user is displayed with the registration page again. If the username does not yet exist, then the username/password is added to the users object as a username : password pair. The fs module is used to re-write the users.json file in order to allow a persistent storage of usernames and passwords, the fs module is also then used to create the users private directory where all of the images will be stored. i.e. `./protectedPhotos/username`



**Login**

Please login to access your picture archive,  
or [register a new account](#)

Username:

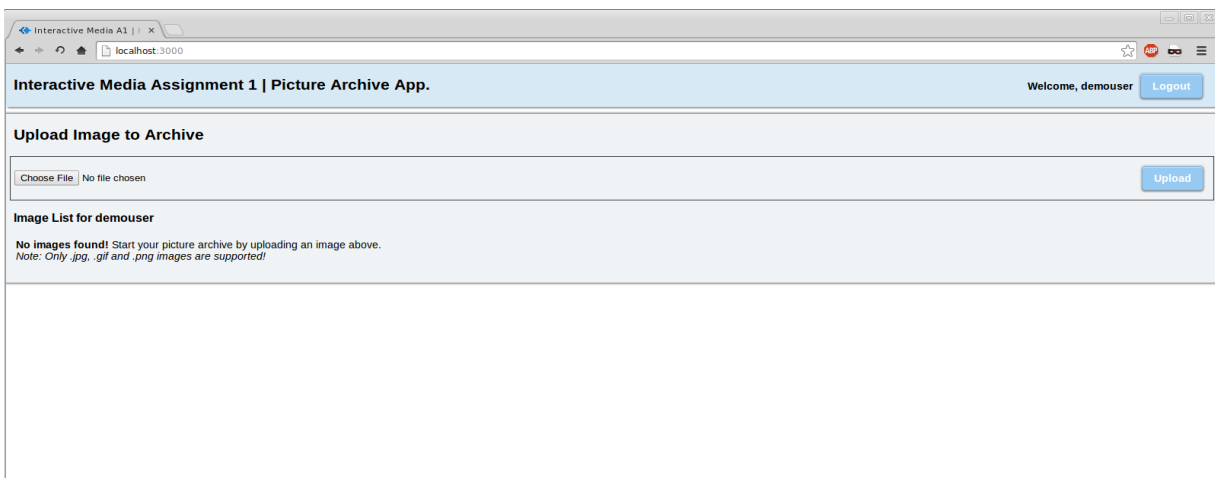
Password:

When a user logs in, the provided username and password is cross-referenced with the users.json file trying to match the username : password pair. If a match is found, then we can assign the current username to the session, which will be used in subsequent methods to identify and store images for a particular user.

The logout method simply destroys the session's username variable and redirects the user back to the login page.

## Image upload and delete methods

A new user is initially presented with a rendered html template as seen below:



Interactive Media Assignment 1 | Picture Archive App. Welcome, demouser

---

**Upload Image to Archive**

No file chosen

---

**Image List for demouser**

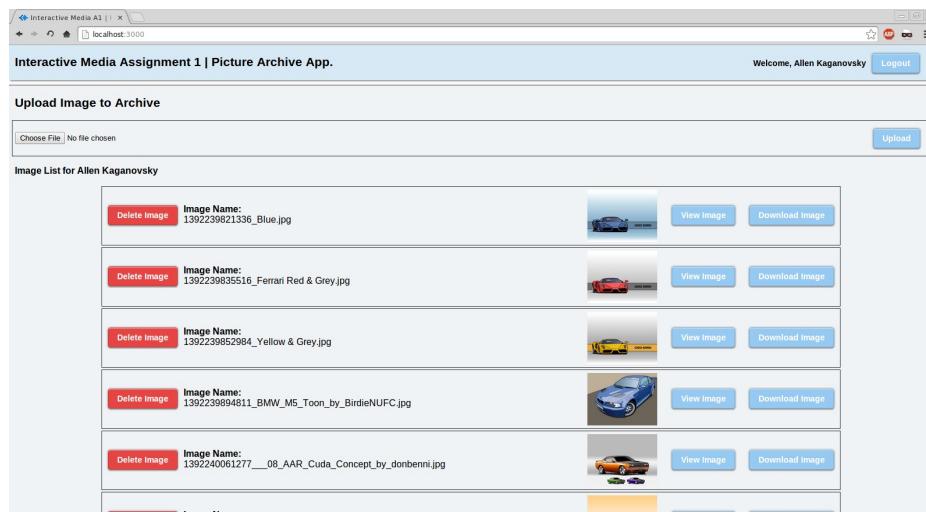
No images found! Start your picture archive by uploading an image above.  
Note: Only .jpg, .gif and .png images are supported!

users are able to upload image files in the formats of .jpg, .png or .gif into the system. When a user initially uploads an image using the form encoded as multipart/form-data, the file is uploaded into a temporary directory './tempUpload'. If the file ends in one of the allowed extensions (.jpg, .png or .gif) then a timestamp to prevent duplicate names is prefixed to the filename and is moved into the users image directory (./protectedPhotos/username). Files that do not end with a .jpg, .png or .gif extension are simply deleted from the './tempUpload' directory.

In order to delete a users image that was previously uploaded, or invalid files uploaded by the user from the temp directory, the fs.unlink method is used to delete the file from the directory.

## Homepage image archive list

When a user logs in, they are displayed with an ejs template of the homepage. This template takes in several variables, one of which is an array of images. This array is populated before the template is rendered to the user by using the fs module and reading all the files within the users specific image directory. If the array of images is empty, the user has no images uploaded and the user is displayed with a page similar to the one above. On the other hand, if the user already uploaded several images, then the images array is populated with the image files specific to the user that is logged in. The template is then able to read the images array and for each image within the array, display a div block with details and a preview of the image.



From the home page the user is able to upload more images into their archive, delete individual images which can be identified using the name or preview image, view a larger image or download the archived image back to their computer.

To re-download a previously uploaded image, I used an easy implementation presented within HTML5 which includes adding the download

parameter into the `<a href='imgurl'>` tag of a link as follows: `<a href='imgurl' download>`

To delete an image, the browser attempts to GET `/delete/imageToDelete`. NodeJS is then able to perform the unlink command using the fs module and redirect the user back to their image archive homepage.

**A live demo is available at:** <http://www.kaganovsky.ca:3000>

You may register a new account, or use *demouser* / *demopass* to login without registering.