# Filesystem and MongoDB Exercises

## Meme DB

### 1.1 Handle all required get requests

Handle all get request to the following paths '**/viewAll'**, '/**addMeme'**, '/**getDetails'**. For each request select and return to the client the appropriate html file, from **views** project folder.

The html for the '/**viewAll'** and '/**getDetails'** response should be **dynamically generated**.

For displaying all memes in **'viewAll**.**html'** the following line should be replaced:

<div id="replaceMe">{{replaceMe}}</div>

For **each** meme:

`<div class="meme">

<a href="/getDetails?id=${meme.id}">

<img class="memePoster" src="${meme.memeSrc}"/>

</div>`

The memes should be sorted by date.

For displaying the details view in **'details**.**html'** the following line should be replaced:

<div id="replaceMe">{{replaceMe}}</div>

With:

`<div class="content">

<img src="${targetedMeme.memeSrc}" alt=""/>

<h3>Title ${targetedMeme.title}</h3>

<p> ${targetedMeme.description}</p>

<button><a href="${targetedMeme.posterSrc}">Download Meme</a></button>

</div>`

### 1.2 Implement upload image functionality

Implement upload functionality, using streams (you may use any **body parser**), so that the app can work with real images (not url addresses), for this task you **must** use the provided form in **'addMeme.html'**

Hints: For parsing the incoming data you might try using **'formidable'** and ' busboy'.

### 1.3 Generate meme objects

For each uploaded meme **create** new object with the following properties:

Meme = {

id: id,

title: title,

memeSrc: memeSrc,

description: description,

privacy: privacy,

dateStamp: Date.now()

}

The id and the filename in the memeSrc should be unique **random** generated strings.

Example of memeSrc: memeSrc:'./public/memeStorage/1/{random string}.jpg'

HINT: For generating the strings you may use the **'shortid'** package or any kind of random string generator

### 1.3 Distribute the files trough folders

Implement some kind of logic, that separates the uploaded images for memes in different folders. For example, if you want to separate the images in such a way, that each folder holds up to 1000 images, when the server receives the 1001 image, it should create new folder, that will hold another 1000 images and so on.

### 1.4 Add download functionality

In the details view, add functionality that allows the user to download the viewed meme image.

### 1.5 On creation implement public/private property

When adding meme to the DB the user should be able to **choose** between adding **public** or **private** entry.

The public entries should be visible in the **'viewAll** ' view and the private ones should be accessible only **trough** request that **specifies** the **ID** of the entry.

Example: /**getDetails?id={{id of the private meme}}**

## Try to add GZIP on the response /optional/

Try to pipe it and provide correct browser headers.

## Deploy the app to the cloud /optional/

Keep in mind that **Heroku** file uploads required third party service. If you want, use some, otherwise the uploaded files may get deleted by their servers (which is fine for this exercise).

## Install MondoDB and Mongoose

Open the provided skeleton and create **connection** with **MongoDB** on its default port-27017. If there are any error messages, about deprecated functionalities, try to fix them.

## Implement the following logic in MongoDB playground

### Create mongoose Schemas

In the appropriate folder create two **Schemas** with names tag and image, which you will be using for **creating**, **reading** and **deleting** data. Each **tag** should have **name**, **creation** **date** and array of **images**. Each **image** should have **URL**, **creation date**, **description** and multiple **tags**. Try to add a function in the tag model that turns **tagName** to **lower case.**

### Create Add functionality

In folder **'handlers'**, implement create functionality in **'tagHandler.js'** for creating new **tags using the provided form.** After you have successfully saved a tag in your DB,

the part: <div class='replaceMe'></div>

should be replaced with appropriate string

for each tag: <div class='tag' id=${tag.\_id}">${tag.tagName}</div>.

In folder **' handlers '** implement create functionality in '**imageHandler.js**' for creating new images, that holds a URL to a picture, description, title and array of tags.

|  |
| --- |
| instanodeDb.saveImage({ url: 'https://i.ytimg.com/vi/tntOCGkgt98/maxresdefault.jpg', description: 'such cat much wow', tags: ['cat', 'kitty', 'cute','catstagram'] }) |

### Create Delete image functionality

**Go to point 3.1**

Create functionality that activates upon clicking the delete button in **results** view.

## Implement search functionality

### 3.1 Search with no parameters

The module should return all data from the DB using ‘**results.html**',

replacing: '<div class='replaceMe'></div>'

for each instance of image with:

`<fieldset id => <legend>${image.imageTitle}:</legend>

<img src="${image.imageUrl}">

</img><p>${image.description}<p/>

<button onclick='location.href="/delete?id=${image.\_id}"'class='deleteBtn'>Delete

</button>

</fieldset>`

### 3.2 Search by tag

The module should return all images holding the selected tag sorted by their date of creation in descending order:

|  |
| --- |
| instanodeDb.findByTag('cat') |

### 3.2 Add search by creation date

The module should return all **images** with their tags **between** **two dates**. The two dates are **optional**. For example - if the **'after'** parameter is missing, take into account only the **'before'** one. If **both** are **missing** **do** **not** **filter** by date. Take no more than the provided number of total results. If 'results' is missing, take no more than 10 results:

|  |
| --- |
| instanodeDb.filter({after: minDate, before: maxDate, results: 24}) |

## Deploy

Deploy your application using [Heroku](http://heroku.com) and [Mlab](https://mlab.com).