Introduction

The Image Portal API is designed to facilitate communication between the web UI and the MongoDB database, where all the data is stored.

Most of the information in this page is also conveyed in a different format, but more interactively, on my swagger site:

https://app.swaggerhub.com/apis/mcarrington1/portal/1.0.0

This site includes the following information:

- Every endpoint
- Models used in conjunction with the end points
- Example API calls
- Example responses

Any data that is highlighted in blue is part of a stretch feature. Anything dealing with the /user endpoint specifically is considered a stretch feature.

API Endpoints

/login

Usage

GET – Pass the username and password as form parameters “in the clear” to access account

Example HTTP call

```
login?username=username-param&password=password-param
```

HTTP Responses

200 – Provides successful operation message

400 – Invalid input provided

Implementation

This will validate against a clear-text password in MongoDB that is presently hardcoded. This is an MVP solution.

/album

Usage

POST – Create an album object; used with a JSON payload

Example HTTP call

```
[
  {
    "id": 1,
    "name": "Swim meet St. Louis Mar 2021",
    "description": "St. Peters RecPlex Swimming",
    "created": "2021-03-15T01:44:51.756Z",
    "images": []
  }
]
```

HTTP Responses

200 – Provides a successful operation message, no other body content

Implementation

This will create a resource under the user document in MongoDB

/album/{albumId}

Usage

GET – Retrieve an album ID object in JSON; where {albumId} is the id key

DELETE – Remove an album, given an {albumId}

HTTP Responses

400 – “Invalid ID Supplied”

404 – “Album not found”

200 – Provides a successful operation with JSON body content

```
  {
    "id": 1,
    "name": "Swim meet St. Louis Mar 2021",
    "description": "St. Peters RecPlex Swimming",
    "created": "2021-03-15T01:44:51.756Z",
    "images": []
  }
```

Implementation

Retrieve or delete an album object from MongoDB with images array

/album/{albumId}/image/

Usage

POST – Post an image to album, given a valid albumID

Example HTTP Call

```
  {
    "id": 1,
    "name": "12312312.png",
    "tag": "ID-123145",
    "location": "https://s3.amazonaws.com/TEST/f82dd4ac221b.png"
  }
```

HTTP Responses

404 – “Album not found”

200 – Will echo back in JSON what was posted.

Implementation

Retrieve an album object from MongoDB with images array

/album/{albumId}/images/

Usage

GET – This will retrieve all images from a given albumId

HTTP Responses

400 – “Invalid id or input”

404 – “Album not found”

200 – Will retrieve info about the images

```
{
  "id": 1,
  "name": "12312312.png",
  "tag": "ID-123145",
  "location": "https://s3.amazonaws.com/TEST/f82dd4ac221b.png"
},
{
  "id": 2,
  "name": "12312315.png",
  "tag": "ID-123199",
  "location": "https://s3.amazonaws.com/TEST/f82dd4ac100b.png"
}
```

Implementation

Retrieve image information from the database

/album/{albumId}/image/{imageId}

Usage

GET – Retrieve an image from an album

PUT – Update an existing image

DELETE – Remove an image from an album

Example HTTP Call

PUT Operation

```
{
  "id": 2,
  "name": "12312315.png",
  "tag": "ID-123199",
  "location": "https://s3.amazonaws.com/TEST/f82dd4ac100b.png"
}
```

HTTP Responses

400 – “Invalid ID or payload provided”

404 – “Album or image not found”

200 – Will echo back in JSON what was posted.

Implementation

This will perform various operations on the images, including updates and deletes

/albums

Usage

GET – Retrieve an array of all albums

HTTP Responses

200 – Provides a successful operation with JSON body content

```
[
  {
    "id": 1,
    "name": "Swim meet St. Louis Mar 2021",
    "description": "St. Peters RecPlex Swimming",
    "created": "2021-03-15T01:44:51.756+0000",
    "images": []
  },
  {
    "id": 2,
    "name": "Swim meet St. Louis Jan 2021",
    "description": "St. Peters RecPlex Swimming",
    "created": "2021-01-15T01:44:51.756+0000",
    "images": []
  }
]
```

Implementation

This will retrieve the data from MongoDB, including underlying image references.

/share

Usage

POST – Send in JSON an email request. Takes an array of imageUrls and email address to send to.

Example HTTP call

```
    "imageUrls": [
     "string"
  ],
  "email": "string"
```

HTTP Responses

200 – Echoes back the body that was sent as confirmation

400 – "Invalid input provided"

Implementation

Internally this will split apart this array and format as HTML, then likely use an "off the shelf" module (e.g. nodemailer, Spring email) to send e-mails with the address.

/user

Usage

GET – Retrieves the existing user's information that is logged in. Can be used for things like displaying text, embedding in email forms, and displaying user's icon

POST – Update existing user info, such as bio, name, and user icon.

Example HTTP Call

```
[
  {
    "userName": "mcarrington",
    "password": "passwordpassword",
    "firstName": "M.J.",
    "lastName": "Carrington",
    "location": "St. Louis, MO",
    "bio": "I do computer-things.",
    "twitter": "http://www.twitter.com",
    "linkedin": "http://www.linkedin.com",
    "personalSite": "http://www.google.com",
    "userIcon": "https://s3.amazonaws.com/TEST/f82dd4ac221b.png",
    "shareAbleData": [
      "twitter",
      true
    ]
  }
]
```

HTTP Responses

200 – Echoes back the body that was sent as confirmation

400 – "Invalid input provided"

Page / User Action to API Mapping

Each Image Portal has a clearly defined set of API endpoints that are called for various on-page operations. Refer to the wireframes for more in-depth UI information.

Login Page

User Action	Endpoint	Operation	Notes
Login to Site	/user/login	GET	This is going to be a simple mechanism and have no real authentication

Account Page

User Action	Endpoint	Operation	Notes
View Existing Info	/user/	GET	This will pull the existing user info to display
Update Info	/user	PUT	Can update any fields for the user

All Albums Page

User Action	Endpoint	Operation	Notes
View Page	/albums	GET	Fetch album list to extract album names and display on album list
Add Album	/album	POST	POST with name and description; creates an album object
View Album	/album/{albumId}	GET	API will return album object based on link clicked and go to thumbnail view

Thumbnail View Page

User Action	Endpoint	Operation	Notes
View Page	/album/{albumId} /album/{albumId}/images	GET	Retrieve album info and info about all images associated with album
Add New Image	/album/{albumId}/image	POST	Associate and upload new image for album
Share Album	/album/{albumId}/images /share	GET / POST	API will gather list of image urls with first call, then send list with email address to share endpoint for sending
Delete Album	/album/{albumId}	DELETE	Remove album and all images

Image View Page

User Action	Endpoint	Operation	Notes

View Page	/album/{albumId}
	/album/{albumId}/image/{imageId}
		GET	Retrieve album info and details of exact image we're viewing

Tag or Rename Image	/album/{albumId}/image/{imageId}	PUT	Update image based on parameters; changed provided via JSON
Delete Image	/album/{albumId}/image/{imageId}	DELETE	Remove image from album
Share Image	/share	POST	Send across image url from the element currently being viewed and email; via JSON to share endpoint