Trek

AWS S3 Integration

Last Update 2024-07-09

Version 1.0

Table of Contents

Document Information

Revision History

Date	Version	Status	Prepared by	Comments
2024-07-04	1.0	Approved/Internal	Jacob Zhu, Matthew Kang	

Approval

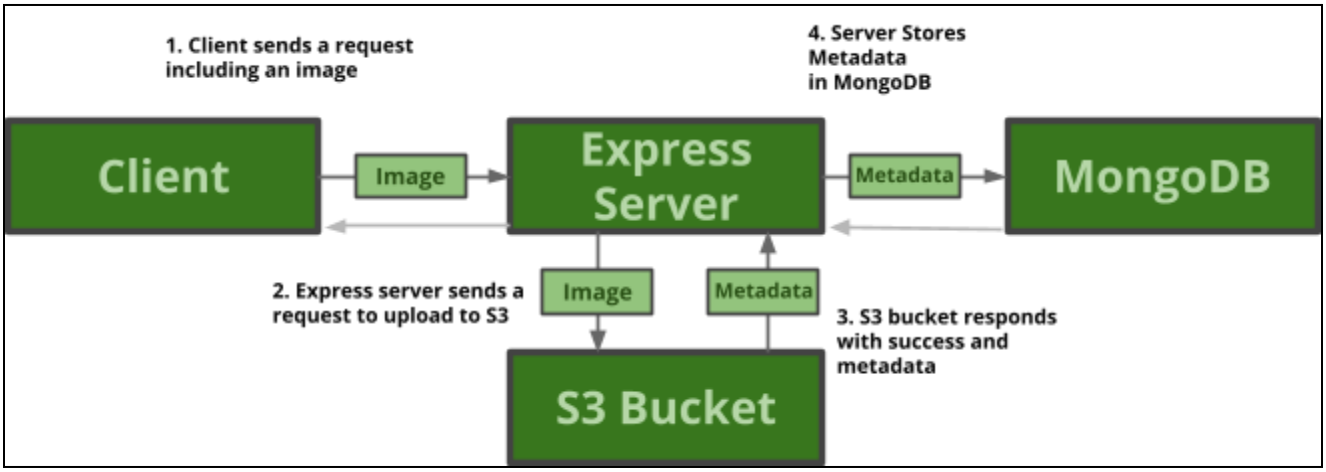
Role	Name	Signature / Initial	Date
Developer	Jacob Zhu	JZ	2024-07-04
Developer	Justin Lieu	JL	2024-07-04
Developer	Kevin Xu	KX	2024-07-04
Developer	Matthew Kang	MK	2024-07-04
Developer	William Xiao	WX	2024-07-04

Trek and S3

AWS S3 is used for storing images and files, including user-submitted images or any images linked from the MongoDB database.

Overview

Uploading to S3 and Logging in MongoDB is handled entirely by the Backend express server for security and simplicity reasons. The diagram below illustrates the flow of data when a user uploads an image.



S3 Pricing

The AWS Free Tier includes:

- 5GB of Amazon S3 storage in the S3 Standard storage class;
- 20,000 GET Requests;
- 2,000 PUT, COPY, POST, or LIST Requests;
- 100 GB of Data Transfer Out each month.

Note that denied requests (4xx) can still count. Private EMPTY S3 Bucket COST ME $1300

AWS Access and API Key

If you have access to the AWS console for S3, you will have access to what files are in S3, and be able to delete them if needed (this may mess up with the MongoDB database).

The IAM user used for uploading is 'trek-s3-user', and any requests from the backend are sent through this IAM user's credential. You will need to put in trek-s3-user's API Keys, both: AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY. These keys grant you to upload to the S3 bucket, but any files uploaded to the bucket are publicly viewable.

Ask Matthew to send over the API Key and put it in your environment file as below:

```
ATLAS_URI="..."
AWS_ACCESS_KEY_ID=your_access_key_id
AWS_SECRET_ACCESS_KEY=your_secret_access_key
AWS_REGION=us-east-2
S3_BUCKET_NAME=cpsc-455-trek
```

MongoDB Schema

The files in the S3 bucket are recorded in the MongoDB Database in the S3Files collection.

S3Files Schema		
Field	**Type**	**Description**
`_id`	ObjectID	Auto-Generated Unique ID for Object
`key`	String	S3 Object Key
`bucket`	String	S3 Bucket Name: 'cpsc-455-trek'
`url`	String	Object Location
`upload_by`	ObjectID	User ID of the User who uploaded
`upload_time`	DateTime	The Time User has uploaded

Backend Validation Before Upload to S3 Bucket

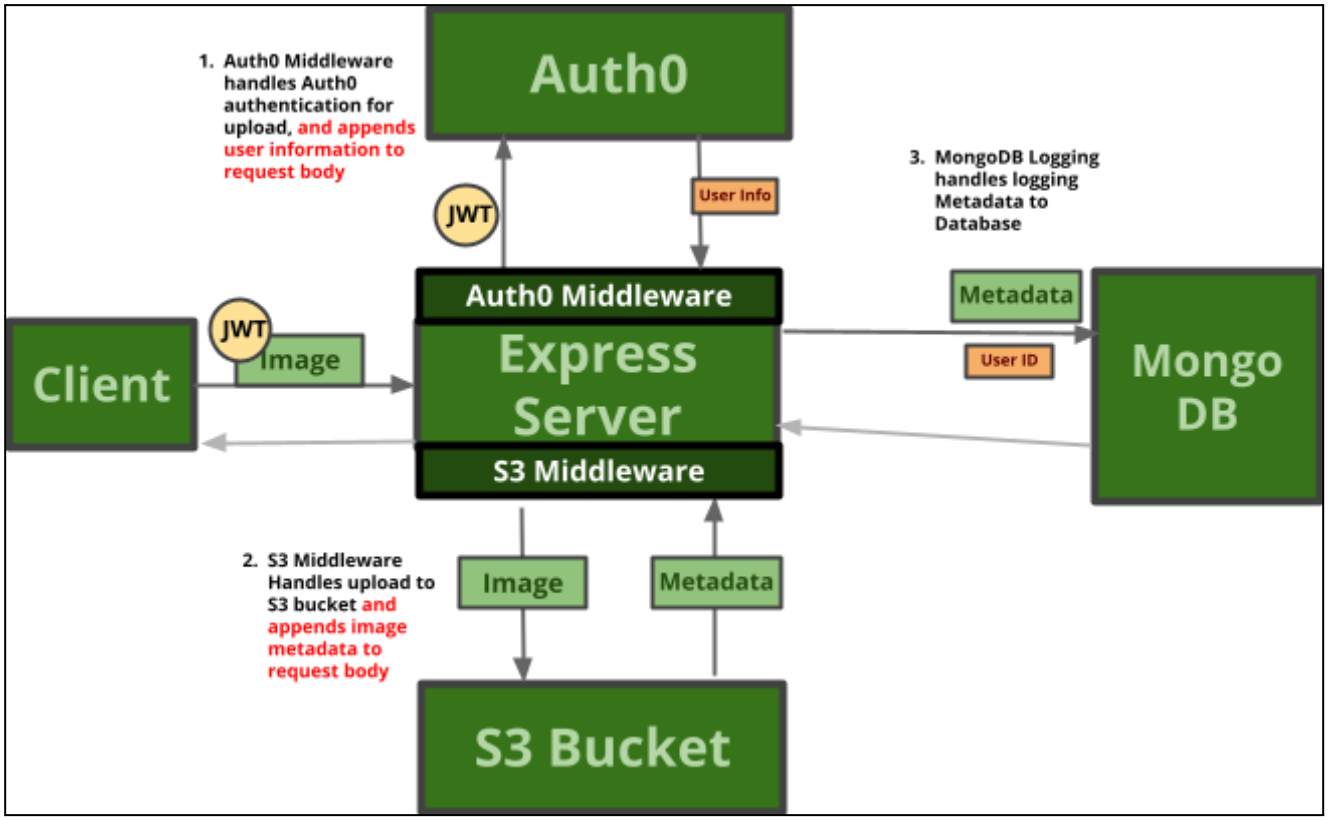
Currently, our S3 bucket does not handle any logic for stopping someone from uploading the entire Wikipedia database. It does not handle restrictions to types of files you can upload. The backend should validate the following.

- Is the user uploading the correct file type
    - Verify case-by-case
- Is the user uploading a sensible size of file
    - Handled in Upload File helper
- Is the user uploading too many files in a period of time
    - Handled in Upload File helper

S3 & MongoDB Middlewares

Express Middlewares work by appending to the request before it gets to the callback function that finally responds.

1. The Auth0 Middleware verifies the JWT with Auth0, then appends the user information received as a result, into the request as `req.user`
2. The S3 Middleware is the next callback function, where it uploads the image to the S3 Bucket, and receives metadata from AWS. It then appends the metadata to the request as `req.files`.
3. The MongoDB Middleware is the next callback function, where it uploads the User ID and file metadata from the previous middlewares into the database.
4. The final callback function responds to the client.



Middleware for Auth0 should be designed by Justin.

Middleware for S3 & MongoDB File logging operations are inside the 'src/s3' directory.

S3 Uploading Middleware

Specify the key in the POST request body in the argument of the upload array, then set the maximum number of photos to be uploaded.

The callback function will append the 'files' property to the Request req that holds information about the file that was successfully uploaded.

```
import { upload } from './s3/upload';

app.post('/test/upload', upload.array('photos', 3), function (err, req, res ) {
    if (err) { // there was an error in upload }

    res.send('Successfully uploaded ' + (req.files?.length ?? 0) + ' files!');

});
```

If the middleware fails at uploading, it will pass an error to the callback function at 'err'.

You can optionally call a nested callback function 'next()' which could be helpful for logging purposes I guess?