Jonathan Lam, Richard Lee, Tiffany Yu, Victor Zhang
Prof. Lusterman
ECE366 — Software Engineering and Large System Design
5 / 13 / 20

Photocol: Final Report

Table of Contents:

Introduction

We present a novel method of handwriting recognition...

...hmmm. Wrong project. Let’s start again.

Photocol is an intuitive website that allows users to upload and view photos, make collections of photos, and share them with other users of the site. The vision is for it to act as a fully-featured photo collection sharing site, with the ability to befriend other users, share and discover collections freely, and be able to view a list of public and discoverable collections personally selected to fit your interests. For this class, we implemented the ability to upload photos, make collections of photos and share collections with other users.

Nontechnical overview of project

(First of all, the name is a portmanteau of “photo” and “protocol”: the former for the subject of the project, and the latter to indicate how we were learning to interface with different web protocols.)

When first arriving at the site, users are prompted to either make an account or login. Upon signing in, users can navigate to a number of pages, namely **Profile**, **Photos**, and **Collections**.

On the **Profile** page, a user can edit his/her profile picture, display name, as well as view Public and Discoverable Collections (see **Collections** below).

On the **Photos** page, a user can view all of his/her photos, neatly displayed in a horizontal masonry layout. The Photos page provides an Upload Photos button, a Select Photos Button, as well as a Delete Photos Button. A user can use the Select Photos Button to toggle Selection Mode. Selected photos can be subsequently deleted using the Delete Button.

Clicking on any photo leads a user to the **Photo** page, which displays an expanded version of that single image, followed by the original upload name, as well as a photo caption and some metadata, such as image dimensions.

On the **Collections** page, a user can create a collection as well as view any collections he/she has access to. Each collection is displayed in a box stating the collection title, its owner, as well as an optional cover image for the collection.

Clicking on any collection will lead a user to the page for that specific collection. The images are once again laid out in a masonry layout. The **Collection** page provides several buttons including, Edit Collection Details (shown only to the owner), Add Photos [to Collection], Select Multiple Photos [from photos currently in Collection], and Remove Photo [from Collection] (the latter three are only shown to the owner and editors). The Edit Collection Details button allows the owner to change the name and description of a collection, as well as add additional users to the collection as viewers, editors, or owners). The Add Photos button opens a modal of the Photos page, and provides the user with the ability to select photos to add to the collection, or upload photos straight from their computer. The Select Multiple Photos button allows a user to select multiple photos and subsequently use the Remove Photos button to remove the photos from the collection.

Collections can be set to Private, Discoverable, and Public. In private mode, only specific users in the Access Control List are allowed to view the collection details, as well as photos. In Discoverable mode, users not on the Access Control List for the collection can see the cover image, title, and description, but not any of the photos. Finally, in Public mode, any user or person who is not signed in can view the cover image, title, description, and collection photos. Discoverable and Public collections can be found on a user's Profile Page.

There is also a complex photo permissions system. As described earlier, collections can be set to public, discoverable, or private. A non-signed-in user can view a photo if it is in a public collection, if it is the cover photo for a discoverable collection, or if it is the profile photo for a user. A signed-in user can view the same photo if any of those conditions apply, if the user is in the access-control-list of a collection which contains the photo, or if the user owns the photo

Discoverable collections may seem somewhat artificial and unnecessary right now (as a user not in its ACL cannot take any action to join the collection), but this was because we hoped to make a larger system of collection discovery (of which a very basic version is implemented on the landing page), where users can send requests to join a collection (similar to how you can request access from a private Google Drive document or Facebook group that you have the link to).

Technical overview of project

Summary

We implemented the backend in three layers (handler, service, and store layers), driven by the Photocol class (initializing services) and the Endpoints class (directing HTTP requests to the handlers and handling authorization and CORS middleware). The handler layer manages basic input validation and packages the inputs into the internal types (User, PhotoCollection, and Photo). The service layer does most of the business logic (passing requests to the store layers as necessary). The store layer uses the DBConnectionManager utility class to make connections to the database (and does so by requesting a connection from the Hikari connection pool). Successful responses are sent back up the call chain to the handler, and serialized into JSON using GSON. Error conditions (including expected and unexpected conditions) are thrown as exceptions and transformed to a standardized JSON response using Java Spark framework's exception mapping, along with an appropriate HTTP status code and additional details, if applicable.

The users-collection and collection-photo relations are many-to-many, so these are stored in junction tables. (The photo-user relation is many-to-one, however, as each photo is directly linked with a particular owner) The database is set up in five tables: one for users, one for collections, one for photos, one serving as the junction table between users and collections (access-control list table), and another serving as the junction table between photos and collections. Foreign keys are implemented to make updates and deletion cascade easily. We use a lot of nested queries and joins to implement the many-to-one and many-to-many relationships.

On the front end, we used React as our main Javascript framework. We use react-redux to store global state (login information) and bind the login state to component props. For API requests, we created a utility class called ApiConnectionManager that manages fetch (AJAX) requests to the backend. Routing was managed using react-router-dom. (Since the backend is running on a different port than the website, proper CORS configuration is set up for all of the JSON endpoints on the backend.) We used Reactstrap (Bootstrap with React bindings) to make our website look more consistent, as well as the Font Awesome icon set. We used the react-photo-gallery library to create a PhotoSelectorList masonry-style photo list display component, which was embedded whenever we needed to display a list of photos (e.g., when listing a user's photos, for each collection, and when selecting a cover photo for a collection or the profile photo for a user).

On deployment, we used docker-compose to set up a network for our containers. We hosted the static website (built using `npm run build` in the React app) on an nginx container, the database in a MariaDB container, and the server (which is packed into an uber-JAR (fatJAR) using maven) on an OpenJDK container. We are currently running this in an EC2 instance for the live demo.

Instructions to run the project are written into the READMEs of the photocol-website, photocol-server, and photocol-DB_SETUP project. The former two are to run the frontend and backend standalone, and the latter is to build and run their respective Docker images.

List of technologies used

Languages: HTML, JS, CSS, (My)SQL, Java

Backend

- Server
  - Java Spark Framework: main HTTP server, streamlines implementation of HTTP protocol, manages cookie-based sessions
  - sl4j: logging service used by the Java Spark Framework
  - GSON: Java implementation of a JSON parser/generator
  - Apache Commons 3: some utility classes
  - Amazon S3 SDK: Java SDK for the S3 service
  - Spring Security Core: used to implement bcrypt for password hashing
  - Metadata extractor: image metadata extractor library
- Server build tools
  - Maven: Java build automation tool and package manager
- Database
  - MariaDB: database technology (drop-in compatible with MySQL)
  - JDBC: Java driver for the database
  - Hikari Connection Pool: for maintaining a connection with the database

Frontend

- Main framework
  - React JS: fully-featured JS framework to handle reactivity and behavior
  - Redux: global state container (mostly used for login information)
  - react-router-dom: easy routing within the app
- Outside libraries
  - Bootstrap: really awesome HTML/CSS/JS library for consistency and responsiveness
  - Reactstrap: React bindings for Bootstrap
  - Font Awesome: really awesome icons
  - react-photo-gallery: nice masonry layout for a list of photos

Deployment

- Containerization:
  - Docker, docker-compose: main container technology
    - adminer: image for database web administration interface
    - mariadb:latest: image for mariadb package
    - nginx:mainline: image for serving the website
    - openjdk:15-jdk-alpine: image for running the packaged server
  - maven-shade plugin: uber-JAR build plugin
- (Ubuntu) Amazon EC2 instance: deployment of Docker images
- Amazon S3: cloud-based storage

Environment

- IntelliJ IDEA Ultimate: IDE
- GitHub (git): version control

Summary of first milestone

The first milestone was focused on implementing the main backend functionality. We began to organize a majority of the major backend endpoints (e.g., endpoints to create users, upload images, and create collections and add images to them). We tested this using a special CLI tool and with curl. Because the project had already reached a stage where persistent storage would be useful, we implemented a MariaDB database. Also, we implemented the S3 SDK. Basic error handling was implemented at this stage.

Summary of second milestone

The second milestone was mainly focused on implementing as much functionality as possible (without focusing too much on quality or design), mostly on the front end. We set up a React webapp and focused on hooking up the major endpoints to the website, and implementing missing endpoints as needed. At the end of this milestone, we had a mostly-working client-facing frontend. We also implemented middleware on the backend and more hierarchical routing endpoints to make routing more neat.

Summary of third milestone

This milestone was mostly about getting the product to be reliable, aesthetically-appealing, complete, and deployable. We implemented Font Awesome and Bootstrap, and spent much time on improving the general aesthetics of the website. There were many UX changes, such as improved error handling (through alerts and form error messages), semantic HTML (such as modals and title attributes), and little tweaks (such as editing the favicon). On the backend, some of the final endpoints were implemented, but most of the work was taken towards reliability (fixing some bugs and adding validation code for user input), extensibility (cleaning up and making code more consistently handle errors, and adding Javadoc-style comments), and security (adding proper salting/hashing for passwords). The database was expanded to include many more fields for users, collections (especially new permissions levels), and photos (especially photo metadata). Public and discoverable photo collections were implemented. The website photo caching policy was fixed so that only images viewable by the currently-logged-in user are accessible. We finished a Docker setup and deployed it to an EC2 instance to host an instance of the website online, and moved values that differed between the deployment and instance and the development instance to environment variables.

Future work

While we were able to accomplish most of our goals and the major functionality of this site, there are still many quality-of-life features (i.e., nice-to-haves) that could be implemented if we were given more time. For example, this could include adding the selected photos on the photos page to a certain collection; searching, filtering, sorting, and pagination features for long lists of collections or photos; more metadata extraction, showing which collections a photo is in; real-time updates when a collection or photo is updated (e.g., through server-sent events). On the technical side, using memcache to have logins persist server restart (right now, restarting the server invalidates all login sessions), finishing up the Docker implementation (including database backups, which are currently a work in progress), and implementing unit testing for the backend and/or the server would be desirable.

Key takeaways

From this project we learned a lot more about dockers as well as learning how to implement different features using react. Learning how to communicate well as a group and work together was another issue that we eventually improved. Creating a website takes a lot of foresight and attention to detail; there are a lot of little features that we consider basic and trivial that sometimes we overlook them are actually essential and make a huge difference. There’s a lot that goes into making a website in both the frontend and the backend, which we were able to experience and learn for ourselves.

Team dynamics

After the second milestone was due, we all realized that we wanted the website to not only be aesthetic but also include many features. There were many late nights getting everything implemented, especially docker. Since it is finals week, we are all busy finishing up the semester and busy with final projects and finals. There were some issues when setting up the docker because the credentials kept accidentally being pushed, which compromised the security and caused an issue. For this milestone we held more meetings and worked together, which prevented a lot of conflicts. Additionally, there were some technical difficulties due to one member’s computer “exploding,” which hindered some progress but we were able to get things to work.

Resources

Github Release:
https://github.com/photocol/photocol-website/releases/tag/v0.3

Server source code:	https://github.com/photocol/photocol-server
Website source code:	https://github.com/photocol/photocol-website
Database setup and dockerfiles:	https://github.com/photocol/photocol-DB_SETUP
Live demo:	http://34.207.93.180/