Term Project Report

ShutterShowcase

COMP 2406 - Fall 2023

Carleton University

Mars Cadieux
101044709

December 12th, 2023

1 Initialization Instructions

1. Navigate to the folder titled “Mars Cadieux Term Project” in your file explorer and open a Node JS terminal in this folder.
2. From your terminal, run the command `npm install` from within the same folder as the file `package.json`. In this case, `package.json` is located in the folder “Mars Cadieux Term Project”.
3. Ensure you have your local MongoDB running (either as a service or add its path in the relevant folder).
4. In your Node JS terminal from within the same folder (“Mars Cadieux Term Project”), run the command `node generator.js`. This will create and populate the database called ShutterShowcase using the data from `photogallery.json`.
5. After `generator.js` has successfully run, you will see all of the new documents printed to the console as well as confirmation messages indicating that all artworks, artists, and patrons have been saved. You may now run the command `node server.js`.
6. Now that the server is listening, you may navigate to `http://localhost:3000` in your web browser. Note that you will be redirected to `http://localhost:3000/login` and you will now see the ShutterShowcase login page.
7. Log in or create an account and have fun! Some pre-existing login credentials are:
   → Username: fox, password: 123 (patron)
   → Username: cat, password: meow (patron)
   → Username: focalpoint, password: artist (artist)
   → Username: bassbeats, password: artist (artist)

2 Youtube Video Link

`https://youtu.be/Mvc9YoJ2h0E`

3 Implementation of RESTful Design

All users have the ability to perform CRUD operations. Patrons and artists can create likes and comments, and artists can create artworks, workshops, and notifications. Users can retrieve artworks using the search feature and they can read data via the ‘artist(s)’, ‘artwork’, and ‘my profile’ pages. Users can delete likes and comments. Users can update an artist’s followers, the number of likes on an artwork, the number of comments on an artwork, and the registered users of a workshop.

I used POST requests to create new objects in the database, and successful creation of these objects always resulted in a status code of 201. I used PUT http requests to update existing objects and these requests result in a status code of 200, or 204 if the object we are updating contains sensitive data (i.e. username and password). I used DELETE requests do delete likes and comments from the database. I use GET requests to retrieve all information to be rendered on each web page, they are responded to with a code of 200 if successful, 404 if not found, and 401 if not authorized.

Input validation is done on the client side unless the nature of the validation absolutely requires use of the database (i.e. checking if a username already exists). If input validation fails on the client side, no request is sent therefore minimizing data transfer.

For the input validation that was done on the server side (login and register requests), the error responses were separated into 404 (not found) when a user attempts to log in with a username that does not exist, 401 (unauthorized) when the user attempts to log in with an incorrect password or if they attempt to register without a password, and 400 (bad request) if the user attempts to register with the username that has already been taken.

The client and server of my web app are completely separate; the client makes requests to the server and the server responds appropriately. The implementation details are hidden from one another, for example when the client requests/sends information it not reference any functions that the server may use.

All requests are stateless; when searching through artworks, the query involves the page number, so only the 10 artworks displayed on the web page are pulled from the database with each request (as opposed to all artworks being pulled and only 10 being shown).

My server data is cacheable and has a time limit (max age) of 1 hour. This decreases data transfer.

Each resource in my web app uses a unique URL that follows an intuitive naming schema (ex. /artists/artistid).

My database is fully integrated in my project, all information about artworks, users, session data, etc is stored in the database and not locally. My system is therefore layered; the client interacts with the server but not the database, and the server queries the database.

4 Critique of Design Choices

A design choice that I made was for my authentication middleware to redirect the user to the login page rather than simply sending an "Unauthorized" response if their session ever becomes logged out. This redirection will also happen if the user manually types in a URL other than `http://localhost:3000` or `http://localhost:3000/login` in the address bar while they are not logged in. I feel this is most similar to what you would see from a real web app, however this redirection does not alert the user of what is happening which I do recognize is a downfall. I did not know of a way to easily alert the user from the server side Without disrupting the redirection.

Another design choice that I made was to use a variable called `isOwnProfile` rather than implementing admin-type authentication to ensure only artists can upload new artworks, etc. This variable was sent to all pug templates and affected which elements were rendered. While this method was concise and effective, it was not the most secure as I had to embed the value of this variable in a hidden span in `artwork.pug` so it could be used in client-side JS. If I had more time I would have implemented a combination of this method and admin-type authentication in my web app.

I was a little unclear on whether users should be able to follow patrons. The specs only mention following artists, so I made a design decision that users should not be able to follow patrons and that patrons do not have a public facing profile. Patrons do not post any artworks or workshops so I felt that there really isn't anything to follow or view.

For my "Browse" page, I made a design decision for the user to only be able to search by one field at a time. I felt that this was more realistic as most public websites don't usually have search tools that involve more than one parameter. In my experience, multi-parameter search tools are usually found in software that is only used internally (staff databases, etc).

If I had had more time I would have made it so that when the user is currently logged in as an artist, the "My Profile" link brings them to their artist page and the page with all of their likes, notifications, etc is more like a tab on that page instead of being a separate page. I also would have given users the ability to upload a profile photo and change their bio.

Extra Functionality

I gave workshops their own page which shows their day/time and a list of all registered users.

I have included two different style sheets and I give users the ability to change the theme of the web app. Note that the theme gets reset to dark mode with each new session so if you log in on the "cute mode" page the web app will switch to dark mode upon logging in. Also please note that the home page is always dark.

I've written several media queries to make the pages of my web app fairly responsive. It's not 100% mobile friendly however it adjusts fairly well when the browser dimensions shrink.

5 Important Notes

I chose to make my web app specific to event photography and I changed the attributes associated with the photos. Instead of artist we have photographer, instead of category we have event, and instead of medium we have venue. These changes were only implemented on the front end; the models, queries, etc all use the keywords "artist", "category", and "medium". I did this so that if you have certain things you need to look for when marking my search page, you can change line 40 of `generator.js` to

```
let gal = require("./gallery/gallery.json");
```

so that my web app uses the original artwork data. Additionally, here is a list of all photographers, events, and venues to help you confirm that my search feature works correctly.

All Photographers:

- Focal Point
- Aperture Addict
- OvrUndr Exposed
- Life on Mars Media
- Diffracted Media
- PixelPulse Media
- ViewFinder Visionaries
- TechnoFrame
- NeonVibe Photo
- VibrantBass Visuals
- MediaMosaic
- GrooveLens Imagery
- BassBeats

All Events:

- Ritual
- Escapade Afterparties
- Escapade Music Festival
- Freak Addiction Krew
- Save the Bass Music Festival
- Signal Events
- White Rabbit
- Intersection

- Sunday Sessions
- Midnight Forever Music Festival
- 'Till Death do us Party
- Undercover presents: Nutek
- Solstice
- DNR Productions

All Venues:

- St Anthony's Banquet Hall
- The 27 Club
- Landsdowne Park
- Cafe Dekcuf
- Delirium
- City at Night
- Collab Space Warehouse
- Club SAW
- Maverick's
- Adelaide Hall