COMP 2406 B - Fall 2022
Term Project – "Open Gallery"

Project Background

The goal of this project is to create a web application to provide exhibition space for artists (to showcase their work) and connect artists with the community. Your application will maintain a database of art items and will support two types of users: patrons and artists. Patrons can browse all the information on the site, add reviews and "likes" to artwork, "follow" artists, and join workshops. Artists will be able to add new artwork or host a workshop. The following sections will outline the minimum requirements of the Open Gallery project. You are encouraged to ask questions to clarify the requirements and constraints of the project.

A note about data

The provided **gallery.json** file contains a sample of artwork data, which you can use to initialize the data for your project. There are 10 different objects with the following format:

```
name: "Artwork name",
artist: "Artist name",
year: "year the artwork was created",
category: "this can be a painting, sculpture, photograph, etc.",
medium: "can be watercolour, acrylic, wood, bronze, silk, etc.",
description: "Extra information about the artwork.",
image: "https://hostname/image.jpg"
```

You must add extra artwork objects. You can add/remove/update the keys and values for each artwork item to serve your project needs. For example, your project can only showcase watercolour paintings. You can use your own art/hobby/sketches/photos or ask your friends to provide theirs. To find classic "artsy" data, please visit the following links for inspiration (Pinterest is also good):

- https://artsandculture.google.com/
- https://www.gallery.ca/collection/search-the-collection

For the purposes of this project, you can assume that artwork and artist names are unique.

You are free to add additional components or functionality to your page. Include a description of any additions/changes you made within your **report.pdf** file.

Term Project – due Friday, December 9, 23:59

Technology Constraints

The server code for your project can be written in JavaScript or TypeScript. All client resources required by your project must be served by your server. Your client must work well within an up-to-date Chrome browser. In some cases, we will run your code to evaluate your project's frontend. Your project's data must be stored using a local MongoDB database (i.e., you cannot use the Atlas cloud database service). You can use **mongoose** to connect to your database. Your final submission must be able to be completely installed using the command **npm install**. Only approved modules may be used. You may assume any modules mentioned in the lectures or notes are allowed. If you are unsure if something is allowed or would like to see if something could be allowed, you should ask for clarification before proceeding. Any additional software, modules, frameworks, etc., you use must be able to be installed using your **npm install** script. You are encouraged to deploy your project to OpenStack.

User Accounts

The application must provide a way for users to create new patron/artist accounts by specifying a username and password. Your account creation page must only require a username and password (i.e., no email, confirm password, etc.) and should not have any security constraints, such as requiring passwords to contain special characters (these are good things to add in a real application). Within your application, usernames must be unique. A user should be able to log in and out of the system using their username and password. Within a single browser instance, only a single user should be able to be logged in at one time (i.e., user A and user B cannot log in to the system within the same browser window). All newly created accounts should be considered **patron** accounts until the user manually "upgrades" themselves to an **artist**. When a user is logged in, they should be able to view and manage information about their account. The application must provide a way for the user to:

1. Change between a patron account and an artist account. If a user changes account types, it should only affect their ability to carry out an action in the future. That is, anything created by a user while they have an artist account should remain unaffected if the user switches back to a patron account.
2. View and manage artists they follow. The user should be able to navigate to the artist's page they have followed. The user should be able to stop following any artist that they have followed.
3. The user should be able to write a review and rate (or simply put "like") for any artwork in the database. You can decide on your rating system. For example, you can rate specifying a score out of 10.
4. View the list of artworks that they have reviewed/liked; choose any artwork in this list and remove their likes/ratings/reviews. They should not be allowed to update/remove reviews or likes by other users.
5. View any notifications they have received about artists that they have followed. For example, if an artist they follow adds a new artwork.
6. Search for artworks by name, artist name, and category keyword, at minimum. Additional types of searches can also be included. The user must be able to navigate to the artwork page for any of the artworks contained in the search results. By default, the search results should show only

10 results and the pagination must be supported (i.e., next/previous buttons to navigate through the remaining search results).

Viewing Artworks

When viewing a specific artwork, a user must be able to:

1. See all the artwork information, including its name, artist, year, category, medium, description, and image. See all the reviews and either rating or the number of "likes".
2. See the artist's name as a hyperlink that allows the user to navigate directly to that artist's page.
3. See each of the category/medium keywords and allow the user to navigate to search results that contain artworks with the same category/medium keyword.
4. Add a review to this artwork, rate it or add "like". This review/rating will be viewable from the user's profile. The user's profile must support allowing the user to remove their reviews/ratings. You may also let them to do so directly from the artwork page.

Viewing Artist Profiles

When viewing the page for a particular artist, the user must be able to:

1. See all the artworks created by the artist. Each artwork entry must allow the user to navigate to that artwork's page.
2. See all the workshops hosted by the artist and enroll in a workshop. At a minimum, there should be a popup notification about the successful enrollment.
3. [*extra functionality] You can allow a user to navigate to the workshop's page, where they can see additional information about the workshop with a list of all the enrolled users.
4. Choose to follow this artist. If a user follows an artist, the user should receive a notification any time a new artwork by this artist is added to the database or a new workshop by this artist is created.

Artists

Every user should be able to switch between patron and artist account types. If a user does not have any artwork in the database but wants to switch to an artist, they should be prompted to add artwork, including the artwork's name, year, category, medium, description, and image. If a user's account type is set to be an artist, the user should be able to do everything a regular user (patron) can do, and also:

1. Navigate to an "Add Artwork" page and add new artwork to the database by specifying all the necessary information such as name, year, category, etc. If the artwork name already exists, the user should not be able to add the new artwork.
2. Navigate to an "Add Workshop" page and add a new workshop by specifying, at minimum, its title.
3. [*extra functionality] You can add other requirements to your "Add Workshop" page, such as the participant's age restriction or prerequisites. For example, if an artist plans a workshop for

kids 8-12 years old, only users within that age category should be able to register. For this to work, each user profile should have DOB information.

Artists should not be allowed to add reviews/likes to their own artworks.

Project Report

You will also be required to submit a **.pdf** project report with your final submission that must include:

1. Detailed steps explaining how to install, initialize, and run your database and server. You must include a database initialization script with your submission that will create a new MongoDB database from your updated **gallery.json** file. This will ensure that a database with the data structure required by your project can be created during the grading process. If you have not used MongoDB, you should include any file resources your server requires to start and load the artwork data.
2. Discussion and critique of your overall design. See the 'Overall Design and Implementation Quality' section of the marking scheme for ideas on what to include in your analysis. You should also consider some of the key concerns of web application development that we have discussed in class, such as scalability, latency, etc.
3. Explanation of any design decisions that you made and description of extra functionality. Examples of extra functionality are provided in this document, but you are not limited to that and are free to add anything to support your design choices.

YouTube Demonstration of Your Application

Provide a link to a YouTube screen capture video with you giving us a demonstration of your project. The video should demonstrate that you have met the project requirements. In the video, you must explain your code (the video must have audio content OR subtitles) and show your code running on diverse correct/incorrect cases.

Make sure the video is "unlisted". If we cannot watch your video because the link is not valid or the video is "private", then no marks will be awarded for the project. You will receive 0 marks for the project if there is no viewable YouTube video link.

Your YouTube demonstration video must not be more than 10 minutes long. It is good if its length is about 5-6 minutes. I will only require the TA's to watch the first 10 minutes. If your video is longer, make sure to demonstrate the essential requirements and use cases within the first 10 minutes.

Code Quality

Your code should be well-written, commented, and easy to understand. This includes providing clear documentation explaining the purpose and functionality of pieces of your code. You should use good variable/function names that make your code easier to read. You should do your best to avoid unnecessary computation and ensure that your code runs smoothly throughout the operation. There should be clear naming of your files and routes.

Term Project – due Friday, December 9, 23:59

Academic Integrity

Submitting not original work for marks is a serious offence. We use special software to detect plagiarism. The consequences of plagiarism vary from grade penalties to expulsion, based on the severity of the offense.

You may:

- Discuss general approaches with course staff and your classmates,
- Show your web page, but not your code,
- Use a search engine / the internet to look up basic HTML, CSS, and JavaScript syntax.

You may not:

- Send or otherwise share your solution code or code snippets with classmates,
- Use code not written by you,
- Use a search engine / the internet to look up approaches to the assignment,
- Use code from previous iterations of the course, unless it was solely written by you,
- Use the internet to find source code or videos that give solutions to the assignment.

If you ever have any questions about what is or is not allowable regarding academic integrity, please do not hesitate to reach out to course staff. We will be happy to answer. Sometimes it is difficult to determine the exact line, but if you cross it the punishment is severe and out of our hands. Any student caught violating academic integrity, whether intentionally or not, will be reported to the Dean and be penalized. Please see Carleton University's Academic Integrity page.

Submit Your Work

Your submission will consist of two parts:

1. Your entire code project for client and server side, including **gallery.json**, **report.pdf**, and any related media or data files. To submit your project, you must **zip** all your files and submit them to the Term Project submission on Brightspace. Name your .zip file "**YourName-project.zip**". Make sure you download your .zip file and check its contents after submitting it. If your .zip file is missing files or corrupt, you will lose marks. Your .zip file should contain all the resources required for your project to run. However, do not include **node-modules** folder.
2. A link to your YouTube video with sound or subtitles demonstrating your project. Provide the link in the **report.pdf** included with your submission.

Project Mark Breakdown

Below is an outline of how marks will be allocated for the term project submission. The final submission will be graded out of 100. The project is worth 15% of the course grade.

Term Project – due Friday, December 9, 23:59

Requirements Met (60 marks)

This portion of marks will be awarded for successfully implementing all (non-extra) functionality listed in sections "User Accounts", "Viewing Artworks", "Viewing Artist Profiles", and "Artists". Following and notification functionality are included as separate items here.

- User Accounts (12 marks)
- Viewing Artworks (12 marks)
- Viewing Artist Profiles (12 marks)
- Artist profile and functionality (12 marks)
- Following and notifications (12 marks)

Note that some features rely on others being implemented. For example, if you have not successfully implemented user accounts, following and notifications would not be able to be evaluated and would not receive any marks.

Extra Functionality (5 marks) (+5 BONUS for extra effort)

You are encouraged to add functionality to your project not discussed in this document. If you do not know where to start, then for 5 marks, you can implement features marked in this document as [*extra functionality]. Alternatively, for 5 marks, you can design and implement other features (of complexity similar to those provided in [*extra functionality]). Additional features, the functionality of greater complexity, or exceptional implementations will be awarded 5 BONUS points.

Interface Quality (10 marks)

Overall interface quality, including factors like organization, visual appeal, responsiveness, and ability to handle different window sizes.

MongoDB integration (15 marks)

For full marks, all artwork, patron/artist user, and session data must be stored in MongoDB. It is advised that you also use Mongoose, but this is not strictly a requirement. You must include a database initialization script with your project submission that will create and add the initial state of your server into an empty Mongo database. This will ensure the data your server requires to function can be generated in order to grade your submission.

Overall Design and Implementation Quality (10 marks)

The overall quality of your system's design and your implementation. This includes things like code quality and organization, effective use of RESTful design principles, use of proper HTTP status codes, proper error handling, use of asynchronous operations, minimization of required data transfer, etc. You must include a short report with your final project submission that explains what good design/implementation properties your project has and whether you implemented something for bonus points. Alternatively, you can discuss this in your YouTube video. You may also explain some possible improvements to the design for partial credit in areas where your design is lacking.