COMP 2406 B - Fall 2023
Term Project – "Open Gallery"

The project should be completed individually. The due date is Friday, December 8, 23:59.
You are required to achieve a grade of at least 50% on the final term project to pass the course.
A grade of less than 50% on the final term project will result in a failing grade for COMP 2406.

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 must maintain a database of art items and support two types of users: patrons and artists. Patrons can browse all the information on the site, add reviews and "likes" to artworks, "follow" artists and join workshops. Artists will be able to add new artwork or host a workshop. Artists and patrons are considered "users", however, their accounts will have different features. 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 21 different objects with the following format:

```
Title: "Artwork Title",
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.",
Poster: "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. There is a lot of "artsy" data available online, but please respect copyright. Instead of copying images for the "Poster" data, provide the links to their sources. If you use your own images/photos, you can upload them to Google Docs or OneDrive, create sharable links, and use the links as "Poster" values.

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

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

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

This application must support two types of registered user accounts:

- Patrons – regular users.
- Artists – artist account can belong to an artist directly or to their agent. An artist agent is any professional who works on behalf of an artist to represent and promote their work.

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.). It should not have any security constraints, such as requiring passwords to contain special characters (these are good things to add in an actual application). Within your application, usernames must be unique. Users 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., users A and B cannot log in to the system within the same browser window). You must use "incognito mode" in the browser to test functionality for several logged-in users. All newly created accounts should be considered **patron** accounts until users manually "upgrade" 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 act in the future. 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 they have followed.
3. The user should be able to write a **review** and add a **like** ♥ for any artwork in the database.
4. View the list of artworks they have reviewed or "liked"; choose any artwork in this list and remove their likes/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 title, artist name, and category keyword, at minimum. Additional types of searches can also be included. For simplicity, make the search case insensitive. The user must be able to navigate (use hyperlinks) 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 title, artist, year, category, medium, description, and poster/image. See all the reviews and 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 and add a "like"❤. This review and the like will be viewable from the user's profile. The user's profile must support allowing the user to remove their reviews/likes. You may also let them 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 users 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 when a new artwork by this artist is added to the database, or a new workshop created by this artist.

Artists

You must create an artist account for every artist in your database. For example, the last seven artworks in the provided **gallery.json** file belong to Banksy. This means that your application must have a corresponding artist account for Banksy.

Every user should be able to switch between patron and artist account types. Suppose a user has no artwork in the database but wants to change to an artist. In that case, they must be prompted to add artwork, including the artwork's title, year, category, medium, description, and poster. Only after a patron successfully adds an artwork does their account upgrade to an artist. 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 title, year, category, etc. If the artwork title 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 or 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. Your name, student ID, project name, course number, and a link to your YouTube video with sound or subtitles demonstrating your project.
2. List of files and their purpose.
3. 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.
4. 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.
5. 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.
6. List of any known errors or control sequences that work not as expected or crash your code.

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 show the functionality of your project on diverse correct/incorrect cases and explain the main parts of your code. The video must have audio content OR subtitles.

Make sure the video is "unlisted". If we cannot watch your video because the link is invalid 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.**

Final Term Project – due Friday, December 8, 23:59

Your YouTube demonstration video must not be more than 10 minutes long. It is good if its length is about 6-7 minutes. I will only require the TAs 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 avoid unnecessary computation and ensure that your code runs smoothly throughout the operation. There should be clear naming of your files and routes.

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 have any questions about what is or is not allowable regarding academic integrity, please contact the 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, intentionally or not, will be reported to the Dean and penalized. Please see Carleton University's Academic Integrity page.

Submit Your Work

Your submission will consist of two parts:

1. Your entire client and server code for the project, 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 20% of the course grade. **You must achieve a grade of at least 50% on the final term project to pass the course. A grade of less than 50% on the final term project will result in a failing grade for COMP 2406.**

Requirements Met (60 marks)

This portion of marks will be awarded for successfully implementing all (non-extra) functionality listed in the 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, functionality of greater complexity, or exceptional implementations will be awarded 5 BONUS points.

Interface Quality (10 marks)

Overall interface quality includes organization, visual appeal, intuitive UI, and responsiveness (the ability to adapt to various window sizes). While the visual design of your web application can be modest, it should be well-organized and feature an intuitive, unambiguous interface. Users should be able to quickly locate all necessary information and control your web application without the need for external documentation.

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 optional. 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 to grade your submission.

Overall Design and Implementation Quality (10 marks)

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

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.

Other

- OpenStack is optional for the project. However, if you are using things we didn't study in class, please install everything, and prepare your project on OpenStack.
- For the project, you can store the username and password information on the server. In general, this information should be kept in data centers.