**Name:** Pak Yin, Kan
**Preferred name (name in Brightspace):** Knox, Kan
**Project name:** Open Gallery
**Student ID:** 101260592
**Course Number:** COMP2406
**Youtube Link for project presentation/explanation:**
https://www.youtube.com/watch?v=SrU6G56qjvU

**List of files and their purpose:**

- server.js: Server code using express for implementation.
- package.json / package.json
- gallery.json: Json file that saves all the default gallery items
- ./views: A list of pug files that would be used for the gallery. (client code)
- ./public: Client side Javascript and css code
- Report.pdf: Final report about the project.

**Detailed steps installing, initializing, and running database and server:**

Before running the code, it should be made sure that the Mongodb has been installed in the local machine. Then install all necessary modules by using "npm install xxxx". "Mongoose, express, pug, http" are used in this project. After that, we can use "node server.js" in terminal on current directory (files containing server.js). Once starting the server and every time when starting the server, it will first check the gallery.json and if the database does not have the data in the json file, it will import it, else the server will just start normally. This ensures that no duplicated data is imported into the server. Once the server is on, go to http://localhost:3000 and we can check the functionalities of the website.

**Design decisions that you made and description of extra functionality**

Other than the basic requirements, I did implement some extra functionalities. For the login page, I added a page for "forgot your password?". This aligns with the modern web design that allows the user to get back their password using their own email address and therefore, for the mongoose schema and when registering for a new account, the user are required to have their email address submitted to the server. If the user wants to get back the password, they can go to the "forgot your password" page and enter their password. It also uses Regex to check if your input is a valid email

or not. Noted that, the actual function of it does not get implemented at this time, so it just checked the validity of the email on client side by using javascript. That is, even if you enter the email, it will not send a real email to the email address.

Other than that, I also implemented an extra functionality that allows the users to navigate to the workshop's page. That is, there is a link in artist profile showing all the workshops they hold and then the user can check each workshop details respectively by clicking the link that navigates to the workshop details page. One thing I think it can be improved is that, maybe I can also add an enroll button in the workshop details page as well since for now, when the user want to add themselves to a workshop, they need to go back to the artist profile and enrolled from there, while the followers got notifications and that navigates to the details page instead.

**Discussion and critique of your overall design**

This project cannot be considered as a "small" project as it needs to support quite a lot of functionalities. As such, scalability became one of the problem when implementing all the functionalities. For example, when first designing/determining the schema for mangoose, it was pretty simple, but when further developing the application, it becomes more and more complicated. To make it even more scalable, we can do the following:

To further enhance the readability and scalability, we can separate the data base initialization script from the server code and use "require()" to call that script instead. This not only enhances the modularity but also can increase the readability. And especially when the application becomes larger and larger, it helps the developer (and their teams) to address and locate the issue for code review.