COMP2406 Term Project

Name: Kelly Huang
Student ID: 101264862

Youtube link: https://youtu.be/you6UvTzjSM

List of files:

- Database folder: to start the Mongo daemon
- **database-initializer.js**: Initializes the database with data from gallery.json. A test user is also created which is set to follow vincentvangogh.
- **Models:**
  - UserModel.js - user schema (for artist and patrons)
  - ArtworkModel.js - artwork schema
  - ReviewModel.js - reviews schema
  - WorkshopModel.js - workshop schema
- **Routers:**
  - server.js - main server file
  - artist-router.js - router that handles requests that has anything to do with a specific artist; including workshops and following feature
  - artwork-router.js - router that handles requests for a specific artwork
- **Views**
  - index.pug: PUG template for home page
  - login.pug: PUG template for login page
  - signup.pug: PUG template for an sign up page
  - dashboard.pug: PUG template for user's dashboard page
  - following.pug: displays all the people the user is currently following
  - artist.pug: PUG template for a **user**'s profile page (not only artists)
  - artwork.pug: PUG template for an artwork's page
  - workshop.pug: PUG template for workshop details page
  - addartwork.pug & addworkshop.pug: adding artwork and workshop respectively
  - notifications.pug: displays user's notifs
  - search.pug: page for searching artworks by artist, title, category
  - searchResult.pug: result page for list of artworks that match the category or medium you clicked on from an artwork page
- Partials
  - header.pug: nav bar
- Server.js: Main server file that includes many miscellaneous routes
  - **Routers**:
    - Artist-router.js: handle all requests to /artist
    - Artwork-route.js: handle all requests to /artwork
- Models/schemas
  - User, artwork, workshop, reviews (more details on these later)
- Static files:

    - Signupclient.js &loginclient.js :clients for the login and signup page
    - Addclient.js: client for the add artwork page
    - Artistclient.js: client for user profile pages
    - Artworkclient.js: client for the artwork pages
    - Dashclient.js: client for the dashboard
    - Styles.css: styles page
- gallery.json : a bunch of artwork data - I added an artist “Takashi Murakami” along with about a dozen of his artworks

Artworks were taken from:
https://www.wikiart.org/en/takashi-murakami/all-works#!#filterName:all-paintings-chronologically,resultType:masonry

—Steps to run program—

1. After unzipping files into desired directory - make sure your current working directory is the one which contains all the files
2. Use the command `npm install` to install all modules/dependencies needed
3. Open up a second terminal window
4. In your second terminal, navigate to the same working directory as the first one and run `mongod –dbpath=database` to start the Mongo Daemon
5. Go back to you first terminal, now initialize the database by running `node database-initializer`
6. After creation - you’ll see “Database initialization complete” - ctrl-c to exit
7. User `node server.js` to run the server - web app will be running at http://127.0.0.1:3000/

—Overall Design and implementation—

Authentication/authorization

There are **4** auth middleware functions

- Auth: to check if it’s the user’s session
- AuthLoggedIn: to check if the user is logged in
    - Need this for following/unfollowing users, loading login/signup/dashboard etc. pages
- AuthArtist: to check if user is an artist account
    - Need this for adding artworks/workshops and loading the respective pages
- SpecialAuth (in the artwork-router): This is to deal with the fact that you have to prompt a patron with no artworks to add an artwork before becoming an artist. Since you can only add artworks when you’re an artist, if the user requests to change account types, a “changeRequest” field is attached to the session(set to true) so we can use specialAuth to bypass the rule.

User dashboard vs. Profile page

When implementing the dashboard page - I was inspired by how most social media apps have a “manage activity” page where you can see all your comments/likes/watch history.

- **Dashboard** - Every user can only access their own dashboard. This is where you:
    - Sign out
    - Change account type
    - If you're an artist: you'll see the option to add artworks or workshops here
    - View and edit your reviews and liked artworks
- **Profile page** - Every user also has a "general" public profile page for other users to view and follow
    - Users can see, but not edit the activity displayed on the profile page

The decision to implement a dashboard page made separating artist and patron functionalities easier and in my opinion it makes more sense to have it this way. I feel it's easier for the user to be able to just navigate to their "hub" where they can manage their account.

Session object

Added fields to session object:

- Username: username of current logged in user
- userId: id of user that's logged in
- loggedIn: true/false value signifying whether a user is logged in or not

Keeping the userId and username in the session object is extremely useful because you can easily access it at all times for queries.

When a user is not logged in…

Users can still browse through all artworks when they are not logged in. Sometimes people have to test out an application before they decide they want to make an account!

When you're not logged in you can't:

- View workshop pages - so random people can't see who's enrolled
- Have access to the following, notifications and dashboard pages - these are patron/artist user exclusive

Database initializer

- All artist passwords from the gallery.json is "password"
- "Test" patron user that's initialized to follow vincentvangogh
- Vincentvangogh is initialized with one workshop

Separating routers

Some routers that handle get requests like /addWorkshop and /addArtwork, were put in the main server.js file since there were already routes that matched that URL pattern in the router files that they would've been in which was causing errors. (Ex. GET /addWorkshop should've been in artist-router.js, but there was already a GET :/username which conflicts with it and it's not worth making a whole other router file)

Workshops

- Workshop names DO NOT have to be unique - which is why I query for workshops by workshop id in my code instead of the name like I do for users and artworks (these two are unique)

Search page

- You have to search artist by username
- Pagination buttons still appear when search results are less than 10 artworks

Extra functionalities

- Dashboard feature
- Users receive notifications whenever someone follows them
- Users receive notifications whenever someone likes their artwork
- Allow users to navigate to a workshop's page where you can see additional information with a list of enrolled users - can also click to navigate to the user's profiles

RESTful design:

**NOT an extensive list of all routers/request - only listing major ones

URL/Pattern: *
HTTP Method: GET
Description: Responding to get requests for URLs that don't exist
Expected Request Body Data: None
Expected Response: Status code of 404 - Page not found
—

URL/Pattern: /artist/{artworkName}
HTTP Method: GET
Description:
Retrieves the artwork with unique title of artworkName if it exists
URL Parameters:

- artworkName - the unique title of the artwork

Expected Request Body Data:
Session
Supported Response Data Types:
• text/html
Expected Response (Success):
• Status code of 200.
• Render an HTML page with artwork's details
—

URL/Pattern: /changeType
HTTP Method: PUT
Description:
Attach a changeRequest field (set to true) to the session object before prompting user to add artwork.
URL Parameters: none
Expected Request Body Data:
Session
Expected Response (Success):
• Status code of 200 - redirect to addArtwork page
—

URL/Pattern: /artist/follow
HTTP Method: POST
Description:
Updates user's following list and sends a notification to the user that they followed
URL Parameters: none
Expected Request Body Data:
A JSON string containing the username of the user to follow
Supported Response Data Types:
• application/json
Expected Response (Success):
• Status code of 200.
—

URL/Pattern: /login
HTTP Method: POST
Description:
Authenticates user to log into their account
URL Parameters: none
Supported Response Data Types:
• text/html
Expected Response (Success):
• Status code of 200 - loads dashboard for user that logged in
—

URL/Pattern: /artwork/addlike
HTTP Method: PUT
Description:
Increments the artwork's number of likes by one and sends a notification to the artist that the user liked it
URL Parameters: none
Expected Request Body Data:

JSON string with artwork name

Supported Response Data Types:

• text/html

Expected Response (Success):

• Status code of 200.

—

Error statuses:

401 - for auth errors (ex. Wrong password, trying to access add workshop as a user)

404 - for not found (ex. Error finding something in the database or requesting a URL that doesn't exist)

409 - conflict (ex. username already in use)

500 - internal server errors (Something went wrong in the server)