Dean Sabbah

101199120

Term Project

COMP 2406

https://youtu.be/qNBPszzlz1Q

Openstack instance: DeanSabbah_Term

Openstack password: DeanSabbah

/*I used games instead of visual art pieces. Art is referred to as games and artists are referred to as publishers. My project should be functionally the same as anyone else's*/

- client
  - scripts
    - gameId.js – Scripts for any game page
    - header.js – Scripts for the header (and by extension, all other pages)
    - login.js – Scripts for the login page
    - newGame.js – Scripts for the game submission page (aka newGame page)
    - newWorkshop.js – Scripts for the workshop submission page (aka newWorkshop page)
    - profile.js – Scripts for any profile page
    - register.js – Scripts for the register page
    - workshop.js – Scripts for any workshop page
  - styles
    - Contains styles for all pages
- data
  - Contains JSON to load into the database
- models
  - Contains Mongoose models
- router
  - auth.js – Router used to log users in and register new profiles. Also used for various checks (Like isPublisher, isLoggedIn…)
  - games.js – Router used to render game pages and the new game page. Also used to post new games, like/unlike games, and post/delete reviews
  - search.js – Router used to render the Search page. Also used to query the database and send the info returned to the user.
  - users.js – Router used to render user pages, and do various checks such as checkFollowing, checkLiked. Also used to send/delete notifications and to follow/unfollow users.

        - workshops.js – Router used to render workshop pages and the new workshop page. Also used to enroll users into workshops, and to check if users are already enrolled.
    - templates
        - pages
            - Contains all full pages used in the app
        - partials
            - Contains the header partial
    - database-initializer.js – Initializes the database. Drops all collections if they exist and then populates the games collection and the users collection with the provided JSON in the data folder
    - db.js – Connects to the database in a way that allows any file in the project to use the connection
    - server.js – Main file. Logs incoming data, renders index page, sets the view engine to pug, adds parsing capabilities and sends requests to the routers. Also creates the server, which listens on port 3000.

Navigate to the root folder of the directory in a command prompt/terminal/shell.

Type “npm install” int your terminal.

After it’s done installing, type “node database-initializer” in the terminal.

When that script is done running, type “node server” in the terminal

In a browser, type “Localhost:3000” in the navigation bar.

Goods:

All routes make sense (anything to do with games is sent down the game route, users to users…).

I use many try-catches to ensure unexpected bugs don’t cause my entire server to crash.

Generally, only send required data.

Bads:

Many of my try-catches cover multiple operations and simply go to error 500, not providing much info to anyone on what the error really was.

Every page change makes at least 3 calls to the server to do various checks. This may be an issue on larger deployments and cause congestion.

I made collections and documents for notifications, workshops and reviews. This is probably not the most efficient way and may cause data to pile up and increase congestion in larger deployments

Putting checks in the header’s script allowed me to not have to rewrite the same check multiple time.

I was going for a retro-y look and feel for the site.

I used some resources from W3School to make the site look better. All instances of this are credited in the code.

Users are able to go to a workshop’s page and see everyone who is enrolled

Using a text index for users and for games made querying the database much simpler.

You can remove likes directly from the game’s page

No **known** bugs nor any control sequences that don’t work as expected