**Database Design and database access/management code**

Database Structure:

Database consists of four tables:

user( \_id: string,   
 first\_name: string,   
 last\_name: string,   
 username: string,   
 password: string,

email: string,  
 admin: boolean)  
 Constraint (username is unique)

User\_id is a unique integer for every user on the site. Admin is a boolean that specifies whether the user is a regular account or an admin account. Password is hashed so that even if the data is accessed, passwords can’t be stolen. Can also contain any amount of o-auth tokens in the format of [name of website]-token.

New entries are added when a new user registers.

artwork( \_id: string,  
 author: string,  
 description: string,  
 media\_url: string,  
 price: float,

purchased: boolean)  
 Constraint (price is 2 decimal places)

Artwork\_id is a unique integer for every artwork on the website. Media\_url is a link to the piece of media itself for display purposes.

New entries are added when a donation is made.

order:( \_id: string,   
 user\_id: string,   
 artwork\_id: string array,   
 price: float)  
 Foreign Key user\_id ⊆ user[user\_id]  
 Foreign Key artwork\_id ⊆ artworks[artwork\_id]  
 Constraint (price is 2 decimal places)

Order\_id is a unique integer ID, user and artwork ID’s refer to the person making the purchase and the list of artwork being purchased. The price is the sum of the prices of each individual artwork in the order, (this will need to be rounded).

An order is created when someone adds an item to the cart and is used to store the data for the cart functionality, and is added to the table when they confirm their purchase, or deleted if they leave the site (or perhaps after an amount of time).

donations( \_id: string,

user\_id: string,  
 artwork\_id: string)  
 Foreign Key user\_id ⊆ user[user\_id]  
 Foreign Key artwork\_id ⊆ artworks[artwork\_id]

Keeps track of who donated what artwork. Created when an artwork is donated. Has an entry for each donated artwork,rather than each donation if multiple pieces can be donated at once.

Entries for all tables can be added and edited manually by admins, but should at least give a warning before confirmation.

Hashing of the users password, and if need be of their access token, will probably be done with bcrypt.js.

Query injection is an issue with MongoDB, but there are publicly available packages like mongo-sanitize that can help somewhat, and by verifying queries, query injection can be avoided.

**Access and Management**:

The Code was designed keeping in mind the idea of each task needing hopefully a single function that the other people in the group could call; logging in is one function, logging out is one function, retrieving a list of users orders is one function, etc. As the project got more complicated this wasn’t always observed, but it was for the most part. These functions are quite strict with what they can take as arguments and how their queries to the database are formatted. Some were written more freeform functions to access the database, where you could give it whatever query you wanted. However, by design at least these functions were only supposed to be called by the specific functions designed to be used by the other members of the group. Several functions in dbController.js call query(), but no part of the code outside dbController.js should need to call query().