Launch instructions

Instance Name: Burak_Yildirim_TermProject
Instance Location: COMP2406-F22
Instance Operating System: Linux-Ubuntu version:22.04

Instance Username: Student
Instance Password: !23qwe
Instance FloatingID: 134.117.128.26

Assignment Folder Pathway: /home/student/Desktop/Assignment5

My work is in the Desktop in a folder called Assignment5

1) Get into the instance with given name password, FloatingID
2) Get into the file called Burak_Yildirim_A5 through the given pathway above
3) Once in the Folder type "npm start" or "node server.js"
4) It will print you a link to localhost:4000 you can use that link get to the website or simply open browser and type the localhost:4000 to reach the website

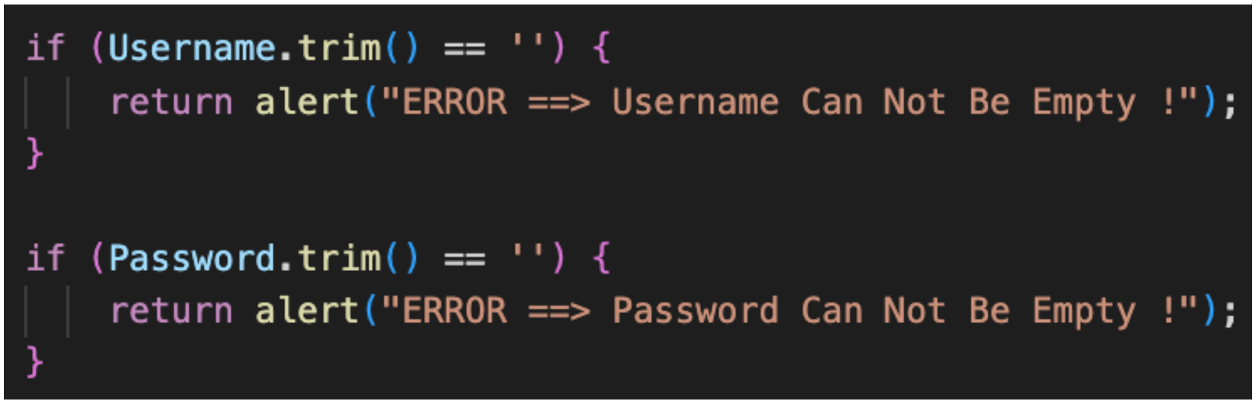
Link To My Video Demonstration: https://youtu.be/n2iHO38kkw4

Overall Design

Generally, if I were to simplify my web project server.js handles the requests between user requests and the data base. All the APIs in the server solves a specific part of the webpage with their correct methods. Database.js is just full of functions that being called by API to manipulate the MongoDB database. And .pug files just renders the frontend the image of the website.

All my API has a proper HTTP status code. If successfully terminates it always returns status code of 200. Also, I have included different response formatting for my API's to be tested with softwires like Postman. All my functions also implemented as asynchronous to save time and has correct implementations of a async/wait.
One great example for this readDataAndInitMongoDB function in database.js. Where I must read from the JSON file to by giving promises with fsPromises just to not skip a line because of async and use wait until fsPromises gets an answer.

The error handling mostly happens in the JavaScript part of the pug files for example when user presses login in the login page in the JavaScript files of the pug where the software checks for an empty value in the HTML id of that text box (login.pug, 92)



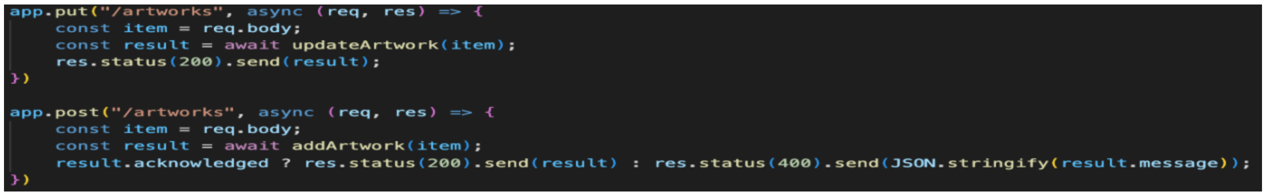
```
if (Username.trim() == '') {
    return alert("ERROR ==> Username Can Not Be Empty !");
}

if (Password.trim() == '') {
    return alert("ERROR ==> Password Can Not Be Empty !");
}
```

In the given photo above you will see I also use trim() function to remove extra space if user just filled textbox with empty space characters this way I can prevent user entering just space charters and also check if one of the textboxes has been empty.

Also, if you watched my video, you will see my website has a very low response rate. And it is just because I have implemented good async/wait functions correctly. This resulted in a quick response.

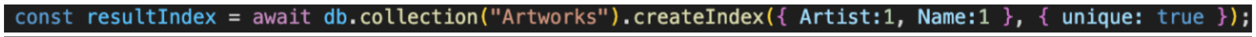
To give an example when we go into the Artwork and when we press add item button it uses POST method because it creates a new item to collection and on the case of an update to Artwork calls put method because we already updating already existing item in the collection (server.js, 103)



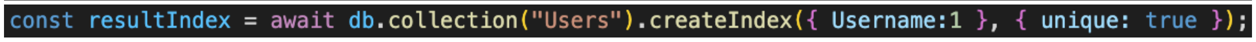
```
app.put("/artworks", async (req, res) => {
    const item = req.body;
    const result = await updateArtwork(item);
    res.status(200).send(result);
})

app.post("/artworks", async (req, res) => {
    const item = req.body;
    const result = await addArtwork(item);
    result.acknowledged ? res.status(200).send(result) : res.status(400).send(JSON.stringify(result.message));
})
```

One of my biggest error handlings is happens when dealing with not allowing user to create duplicates on the couple things. As I mentioned in my video, I solve this by creating unique indexes for the things that I don't want to be duplicated in Collections. And these are the lines of code to create my unique indexes (database.js, 251, 273)

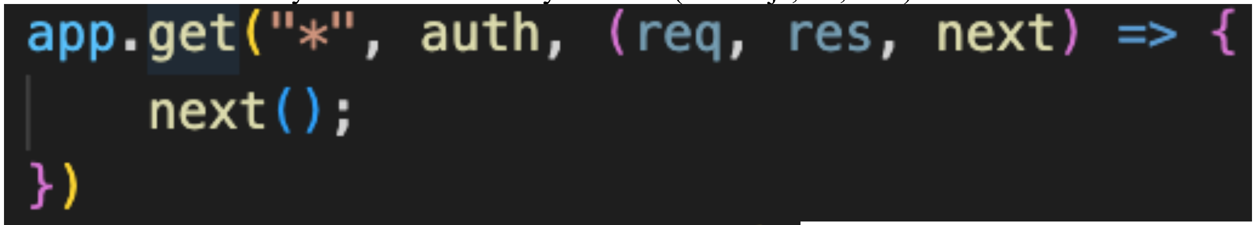


```
const resultIndex = await db.collection("Artworks").createIndex({ Artist:1, Name:1 }, { unique: true });
```

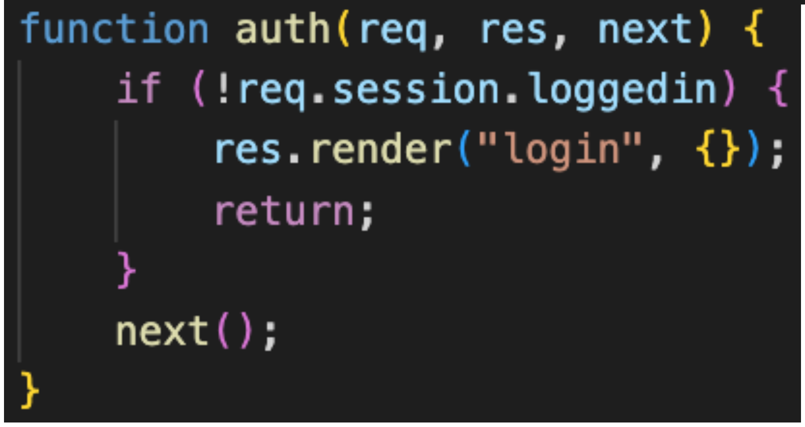


```
const resultIndex = await db.collection("Users").createIndex({ Username:1 }, { unique: true });
```

The most important part of my assignment is the Authentication Middleware that checks every request to API whether user is Authenticated or not otherwise it will kick them to the login page. This allows for 0 anonymous entries on my website (server.js, 44, 234)



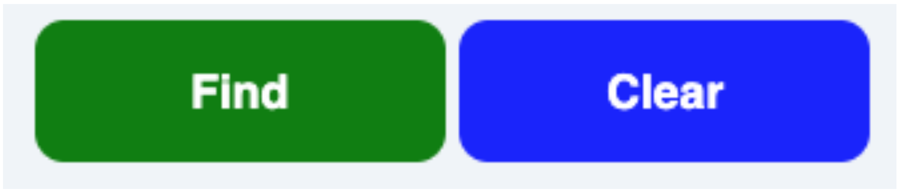
```
app.get("*", auth, (req, res, next) => {
    next();
})
```



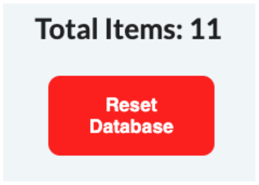
```
function auth(req, res, next) {
    if (!req.session.loggedin) {
        res.render("login", {});
        return;
    }
    next();
}
```

Design Decisions

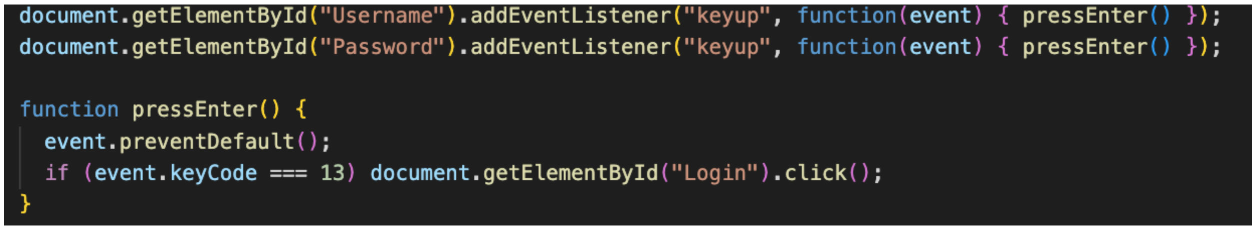
I have implemented small extra things to my project to boost overall user experience in my website
Such as addition of "CLEAR" button that will clear the textboxes. It is also good for a person that will be testing the website also since they must enter bunch of different value it is going to save them a lot of time



Another design implementation is to print the number of Artworks to user separately at the top of the list that way they will see the updates in real time and don't have to count all the items by themselves. Also I have added button in Artworks page to reset the database to its original way. This helps tremendously if wants to do everything from start over. Or I personally used it to delete the extra test cases when I was doing troubleshooting.



Also, I coded in a way that user doesn't require it's mouse as well for example in the login page after user typed all their information they don't have to click on "login" they can just hit enter to login as well (login.pug, 79)



```
document.getElementById("Username").addEventListener("keyup", function(event) { pressEnter() });
document.getElementById("Password").addEventListener("keyup", function(event) { pressEnter() });

function pressEnter() {
  event.preventDefault();
  if (event.keyCode === 13) document.getElementById("Login").click();
}
```

I have also made easier for user to access anything from different pages Example would be if a user is on his user page looking to his likes from there, she can directly go to the Artwork he liked or the Artist he follows. This saves use time and makes it easier for them to navigate in the web page.

Lastly if I were to give advice for improvements of this web application it can have more features and it can build on features such as user system although there are different users in the system, they don't have a unique traits that is attached to their user type.