CS 174 (Server Side Web Programming)

Final Project Report

Professor Ritik Mehta

December 8, 2023

By Justin Yamamoto

Project Overview

The purpose of this project was to implement a web application that supported the following functionalities: User registration, user sign in, supporting creating operations with a MySQL database, and file uploading.

To meet these requirements, I created a basic social media application using MySQL, ExpressJS, ReactJS, and NodeJS. The application was split into two parts: client and server. The server consisted of ExpressJS, NodeJS, and MySQL and created routes to allow users to retrieve / send data to the MySQL database or ReactJS frontend. The client consisted of ReactJS and would make API calls to the server in order to send or retrieve data for the user.

Database Management

Sequelize was used to integrate MySQL with NodeJS and ExpressJS. Using Sequelize, I was able to define models that could be converted to MySQL tables upon running the server. The "Users" model is shown below.

```
//creating a SQL table. create model and export

module.exports = (sequelize, DataTypes) =>{
    //define fields / columns for your table
    const Users = sequelize.define("Users", {
        username:{
            type: DataTypes.STRING,
            allowNull: false,
        },
        password:{
            type: DataTypes.STRING,
            allowNull: false
        }
    }
    )
    return Users;
}
```

All MySQL tables are shown below.

```
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| id        | int          | NO   | PRI | NULL    | auto_increment |
| username  | varchar(255) | NO   |     | NULL    |                |
| password  | varchar(255) | NO   |     | NULL    |                |
| createdAt | datetime     | NO   |     | NULL    |                |
| updatedAt | datetime     | NO   |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
```

Users Table

```
mysql> describe posts;
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| id        | int          | NO   | PRI | NULL    | auto_increment |
| title     | varchar(255) | NO   |     | NULL    |                |
| postText  | varchar(255) | NO   |     | NULL    |                |
| username  | varchar(255) | NO   |     | NULL    |                |
| timestamp | datetime     | NO   |     | NULL    |                |
| imagePath | varchar(255) | YES  |     | NULL    |                |
| createdAt | datetime     | NO   |     | NULL    |                |
| updatedAt | datetime     | NO   |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
```

Posts Table

```
mysql> describe comments;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| id          | int          | NO   | PRI | NULL    | auto_increment |
| commentBody | varchar(255) | NO   |     | NULL    |                |
| username    | varchar(255) | NO   |     | NULL    |                |
| timestamp   | datetime     | NO   |     | NULL    |                |
| createdAt   | datetime     | NO   |     | NULL    |                |
| updatedAt   | datetime     | NO   |     | NULL    |                |
| PostId      | int          | YES  | MUL | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
```

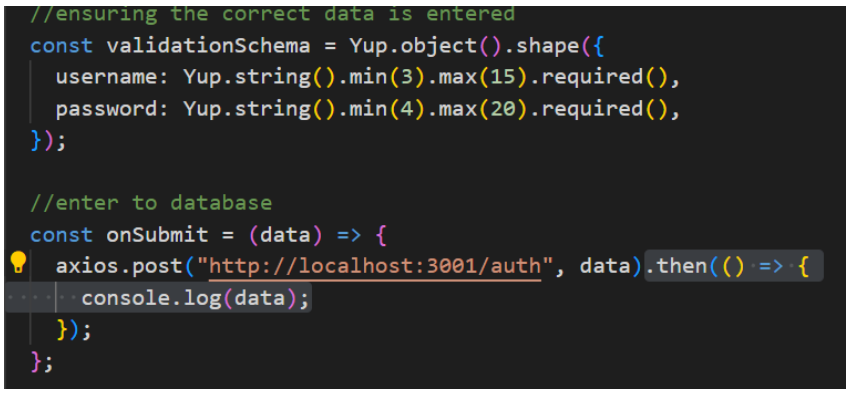
Comments Table

The config.json file in the server specifies the database connection (database to connect to, username, password for access) and is shown below

```
{
  "development": {
    "username": "root",
    "password": "#jySJSU2024",
    "database": "cs174_finalproject",
    "host": "127.0.0.1",
    "dialect": "mysql"
  },
  "test": {
    "username": "root",
    "password": null,
    "database": "database_test",
    "host": "127.0.0.1",
    "dialect": "mysql"
  },
  "production": {
    "username": "root",
    "password": null,
    "database": "database_production",
    "host": "127.0.0.1",
    "dialect": "mysql"
  }
}
```

Registration / Login

The client side code for user registration is shown below



```
//ensuring the correct data is entered
const validationSchema = Yup.object().shape({
  username: Yup.string().min(3).max(15).required(),
  password: Yup.string().min(4).max(20).required(),
});

//enter to database
const onSubmit = (data) => {
  axios.post("http://localhost:3001/auth", data).then(() => {
    console.log(data);
  });
};
```

The form that the user submits has a validation schema which ensures that the user enters appropriate data for their username and password. After submitting the registration form, the entered data is passed to the auth routes in the backend.

The auth route on the server side that handles the data entered by the user is shown below

```
//user registration
router.post("/", async (req, res) =>{
    const {username, password} = req.body;
    //hash the password + salt rounds. take hash result and store
    await bcrypt.hash(password, 10).then((hash) => {
    Users.create({
            username: username,
            password: hash,
     })
     res.json("USER REGISTERED")
    })
});
```

Upon receiving data from the client, the server retrieves the entered username and password using req.body, which represents the form data that the user submitted. The password is then hashed using bcrypt with 10 rounds of salting. This password hash is then used to create a user with the entered username.

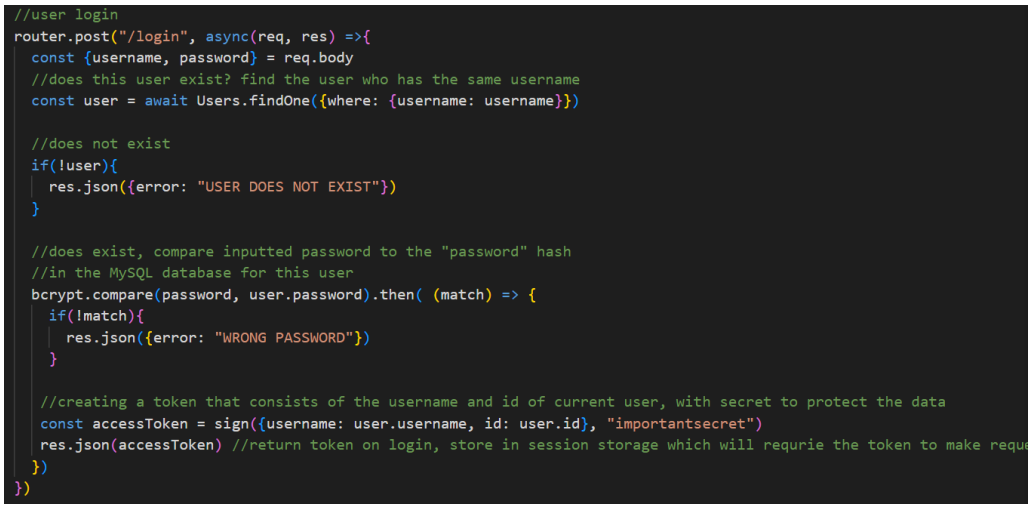
The client side for user login is shown below

```
const [username, setUsername] = useState("");
const [password, setPassword] = useState("");

const login = () => {
  const data = { username: username, password: password };
  axios.post("http://localhost:3001/auth/login", data).then((response) => {
    if(response.data.error){
      alert(response.data.error) //errors in login?
    }
    else{
      sessionStorage.setItem("accessToken", response.data) //storing token in sessionStorage
      console.log(sessionStorage.getItem("accessToken"))
    }
  });
};
```

When the user enters their username and password in the login form, these values are automatically set to the username and password useState variables. This data is then passed to the server in order to verify the user who is trying to login. Upon receiving the response from the

server, the client will either alert the user of a login error if their credentials were incorrect, or set the users access token so they can use the application.

The server side code for user login is shown below



```
//user login
router.post("/login", async(req, res) =>{
  const {username, password} = req.body
  //does this user exist? find the user who has the same username
  const user = await Users.findOne({where: {username: username}})

  //does not exist
  if(!user){
    res.json({error: "USER DOES NOT EXIST"})
  }

  //does exist, compare inputted password to the "password" hash
  //in the MySQL database for this user
  bcrypt.compare(password, user.password).then( (match) => {
    if(!match){
      res.json({error: "WRONG PASSWORD"})
    }

   //creating a token that consists of the username and id of current user, with secret to protect the data
   const accessToken = sign({username: user.username, id: user.id}, "importantsecret")
   res.json(accessToken) //return token on login, store in session storage which will requrie the token to make reque
  })
})
```

Upon receiving the username and password data from the client, a query to the Users table is done to find the username with the specified username. If there is none, then the user is notified that they have entered the wrong username. If a user does exist, bcrypt is used to check if the password meets the value of the user's password hash. If it does, then the password is correct and the user is logged in. If not, the user is told their password is incorrect. Once a user has logged in, an access token is generated from their username and id.

Creating a Post

The client code is shown below

```
const onSubmit = (data) =>{
 console.log("START")
 console.log("ACCESS TOKEN" + sessionStorage.getItem("accessToken"))
 axios.post(
  "http://localhost:3001/posts/",
  data,
  {
    headers:{
      accessToken: sessionStorage.getItem("accessToken"),
    },
  }
 )
 .then( (response) => {
  if(response.data.error){
    console.log(response.data.error)
  }else{
    console.log("SUCCESS")
  }
 })
```

Upon submitting the create post form, the data along with the user access token is passed to the server

The server code is shown below

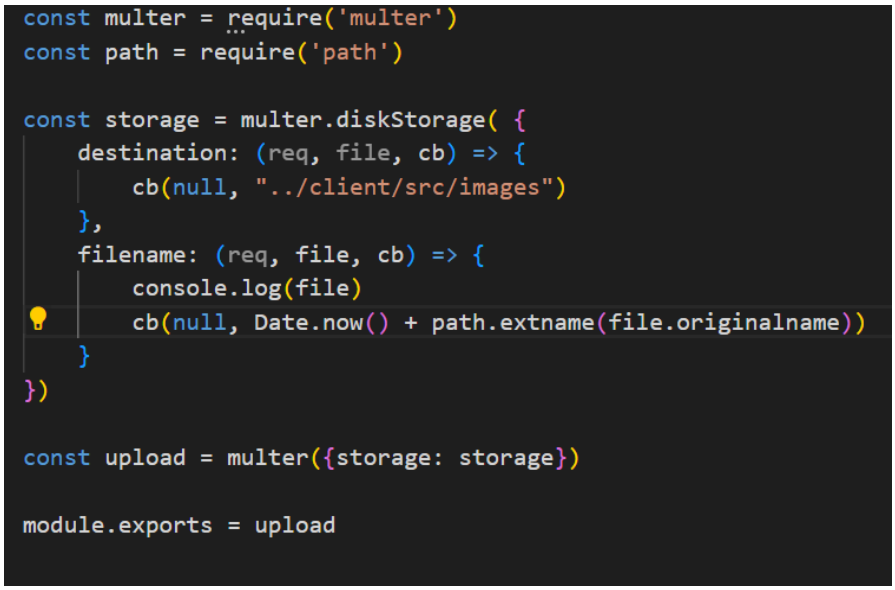
```
//insert into database. HTTP POST
router.post("/", validateToken, upload.single("image"), async (req, res) =>{
 const post = req.body
 post.title = req.body.title
 post.postText = req.body.postText
 post.username = req.user.username
 post.timestamp = Date(Date.now).toLocaleString()
 if(req.file){
  console.log(req.file.path)
  post.imagePath = req.file.path
 }
 await Posts.create(post)
})
```

The server takes the body of the data and makes that most of the post attributes, such as the title and post text. The username is obtained from the validateToken middleware which gets the username from the access token. If a file is detected by the upload middleware, the path to the file is included as well.

Authentication / validateToken middleware. Checks and verifies the access token of the user

```
const {verify} = require('jsonwebtoken')

const validateToken = (req, res, next) => {
    console.log("VALIDATE TOKEN")
    const accessToken = req.headers["accesstoken"]
    console.log(accessToken)

    try{
        console.log("YES")
        const validToken = verify(accessToken, "importantsecret")
        req.user = validToken
        if(validToken){
            return next()
        }

    }catch(error){
        console.log(error)
    }
}

module.exports = {validateToken}
```

File Upload Middleware. Specifies a storage location and naming convention for uploaded files



```
const multer = require('multer')
const path = require('path')

const storage = multer.diskStorage( {
    destination: (req, file, cb) => {
        cb(null, "../client/src/images")
    },
    filename: (req, file, cb) => {
        console.log(file)
        cb(null, Date.now() + path.extname(file.originalname))
    }
})

const upload = multer({storage: storage})

module.exports = upload
```