Final Project Documentation

Jennifer Hua, jthua@iastate.edu

Maddelynne McGovern, mrm4@iastate.edu

Sagnik Dey, sdey@iastate.edu

Iowa State University

SE/ComS319

Construction to User Interfaces

Professor Ali Jannesari

May 7, 2024

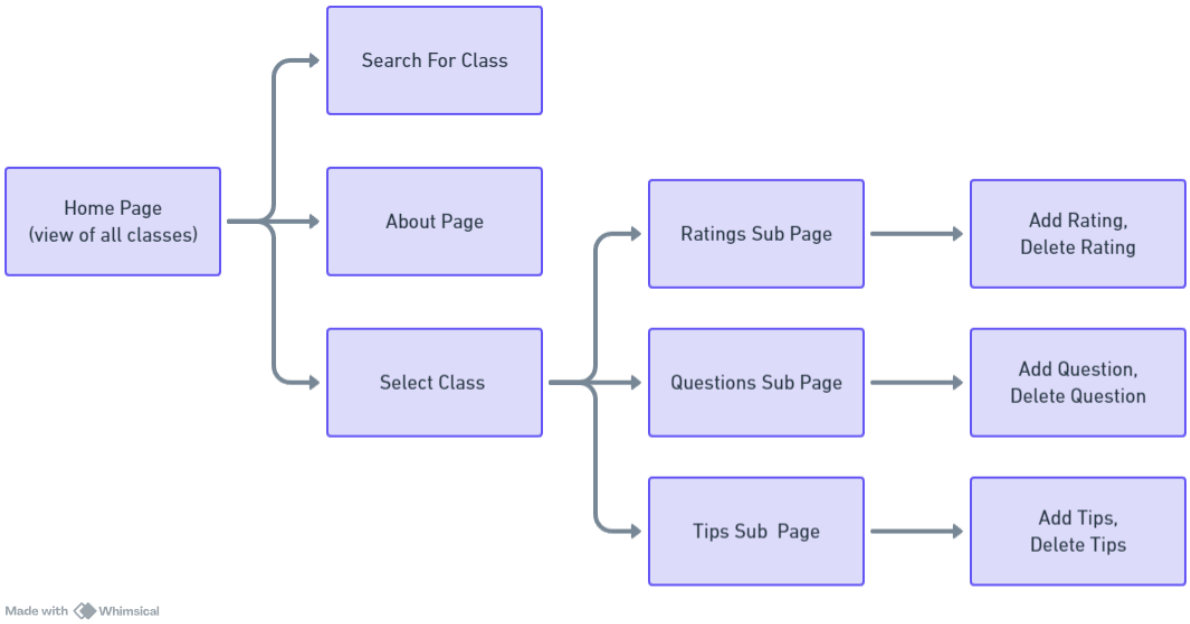
Table of Contents

Project Description

At Iowa State University there are a vast number of classes a student can choose from. Thus, picking classes can be an overwhelming and daunting task, especially for new students. As of right now, students are able to look up classes and read their descriptions or ask their advisors for more information. However, there is no way for students to easily get more in-depth from other students who have taken the class already. While students are able to talk to upperclassmen and get advice about classes, not every student is able to do that. Especially new students who don't know many other students yet. Our project aims to solve this problem by creating a space where students can share their thoughts and advice on classes they have taken. By creating this space, new and current students will benefit and have an easier time choosing their classes. Additionally, this will provide students with many different perspectives from other students that you couldn't get otherwise.

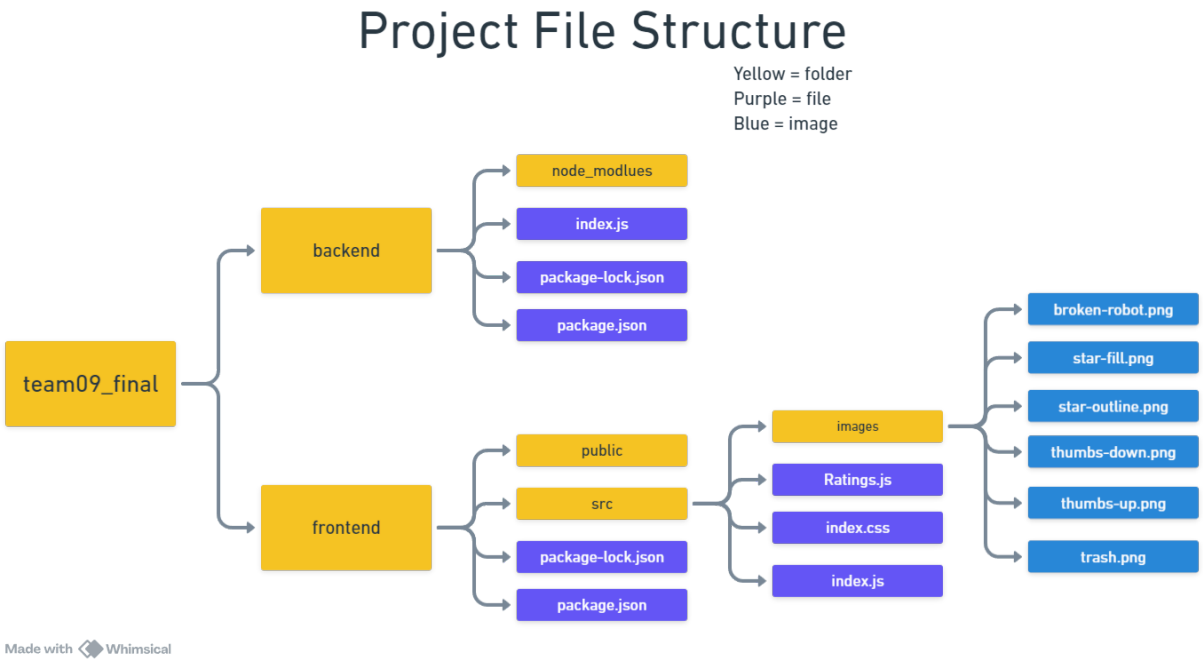
Our project will consist of a home and about pages. The home page will list all of the classes with its description and the amount of posts it has. From there, a user can find the class they're interested in and click on it. When clicked, a user will be taken to a page for that class which includes a ratings section, questions section, and tips section. The ratings section will contain reviews of the class from other students, the tips section contains advice/tips from other students, and the questions section will allow students to leave and answer questions.

Software Functionality Diagram

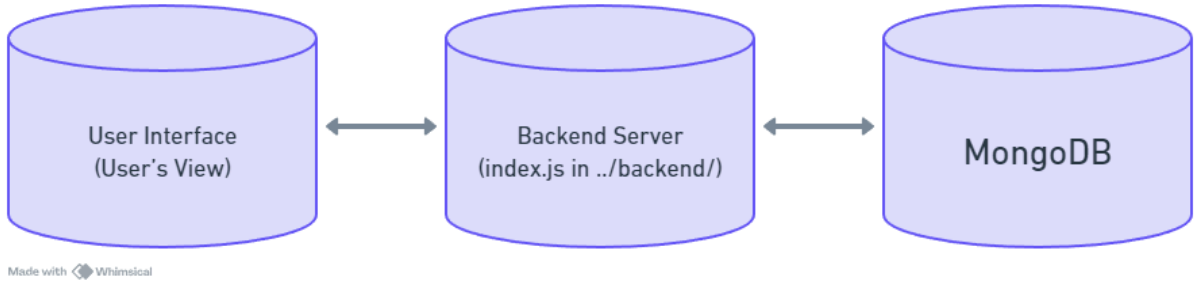


Files and Directory Architecture

Our project contains two main folders, backend and frontend. The backend folder contains our main javascript file ‘index.js’ for the backend. The backend utilizes Express, Node.js, MongoDB, etc. The frontend utilizes React with all of our react components in ‘Ratings.js’. The ‘index.js’ file in the frontend is the main React component that renders our other components. The ‘index.css’ file holds our custom styles for our webpage. Lastly, the images folder holds images for various icons we use in our webpage.

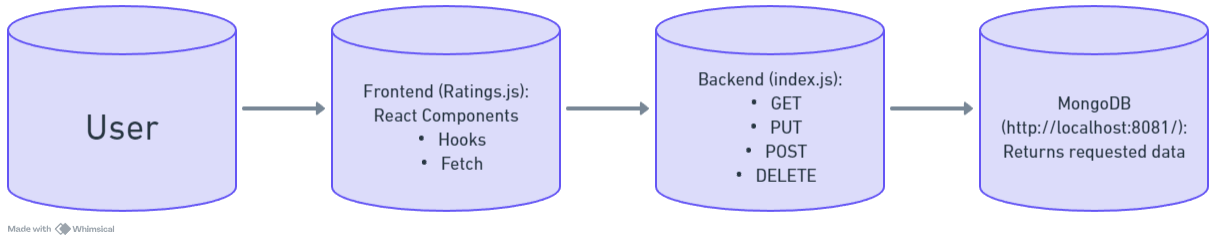


Client – Server Architecture



As the user navigates through our website, they are able to click on various links/buttons such as the links to specific classes, or post their reviews. The user's actions then initiates a fetch in the frontend. This fetch is sent to the backend where the backend now makes a request to the database. After successfully connecting to the database, the database will send back the requested information, which is passed along back to the frontend. Our database collections include: courses, questions, ratings, and tips and we subsequently make fetches at '/courses/', '/questions/', '/ratings/', and '/tips/'.

Logical Architecture



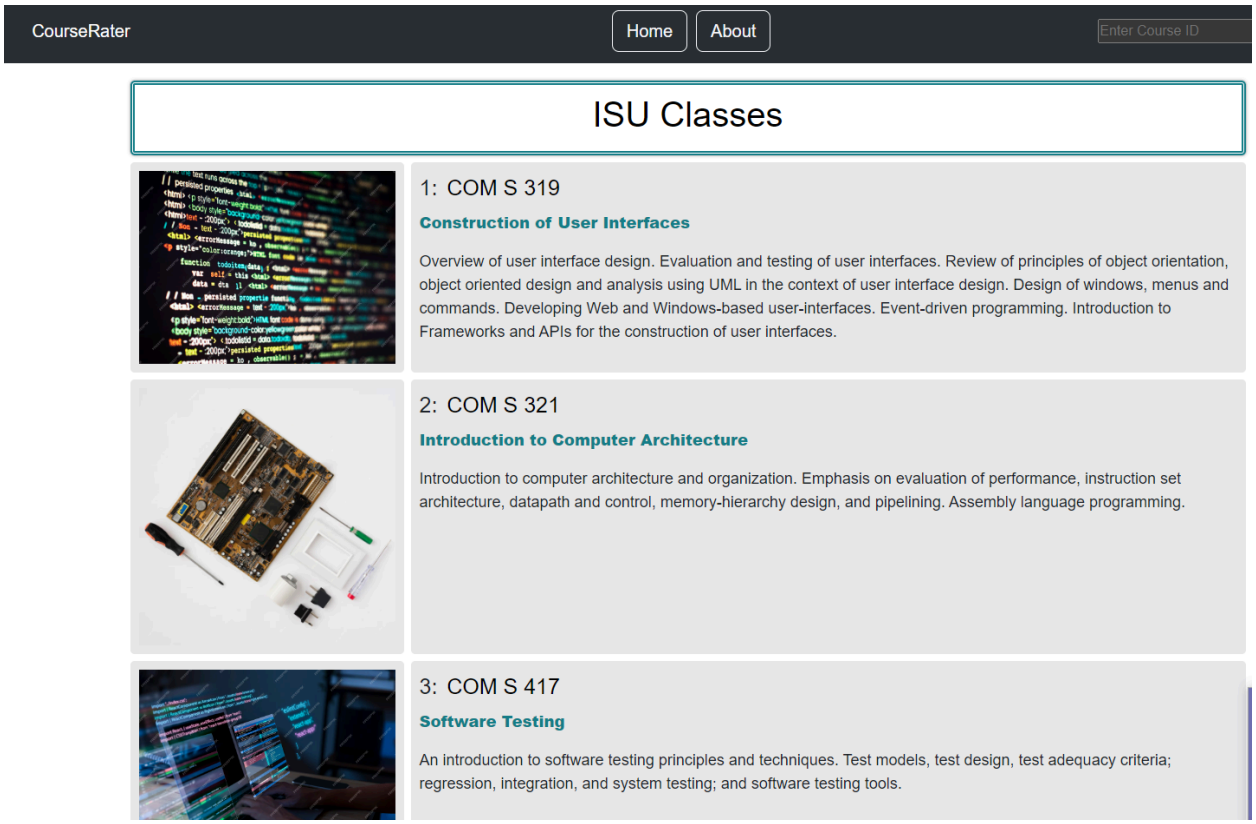
Our website uses a single page design with React, thus when a user navigates to a different page it'll switch to the desired page using React hooks. We also use hooks to update information and communicate with the backend when a user posts/updates a rating/question/tip. The server will make get, put, post, and delete operations to the database when prompted by the user's actions (clicking a button, adding data, etc.). The database will return the requested data and then displayed to the user.

Database and API

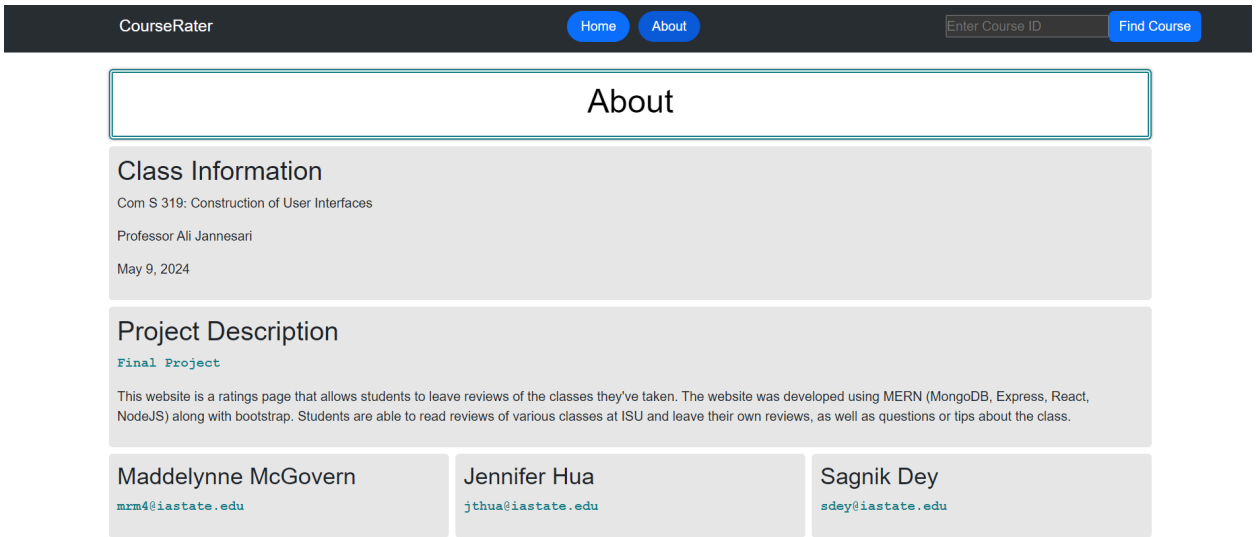
GET	/courses /courses/:id /questions/:id /ratings/:id	The first endpoint will get all courses available in the database to display as a list for users to look through. The rest of the endpoints are as

	/tips/:id	follows: courses indicate all courses with specified id and will only display that specific course. Questions, ratings, and tips indicate all the questions, ratings, and tips posted for the specified course.
PUT	/ratings/helpful /ratings/unhelpful /questions/:id	This updates the amount of thumbs up / thumbs down whenever a user clicks either in the ratings subpage. The questions endpoint updates the replies in the questions sub page.
POST	/courses /ratings /questions /tips	These endpoints add data to their respective collections in the database when a user inputs data and clicks the 'Post' button.
DELETE	/courses/:id /ratings/:id /questions/:id /tips/:id	This deletes user entered data. For example, ratings they posted.

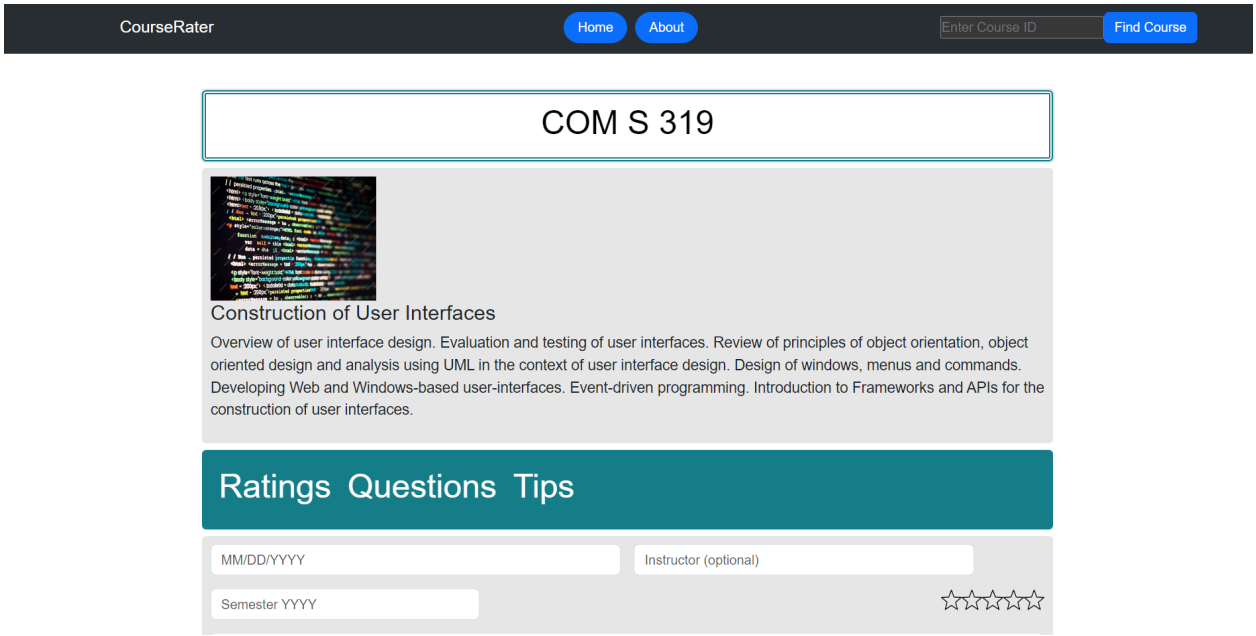
Web Views



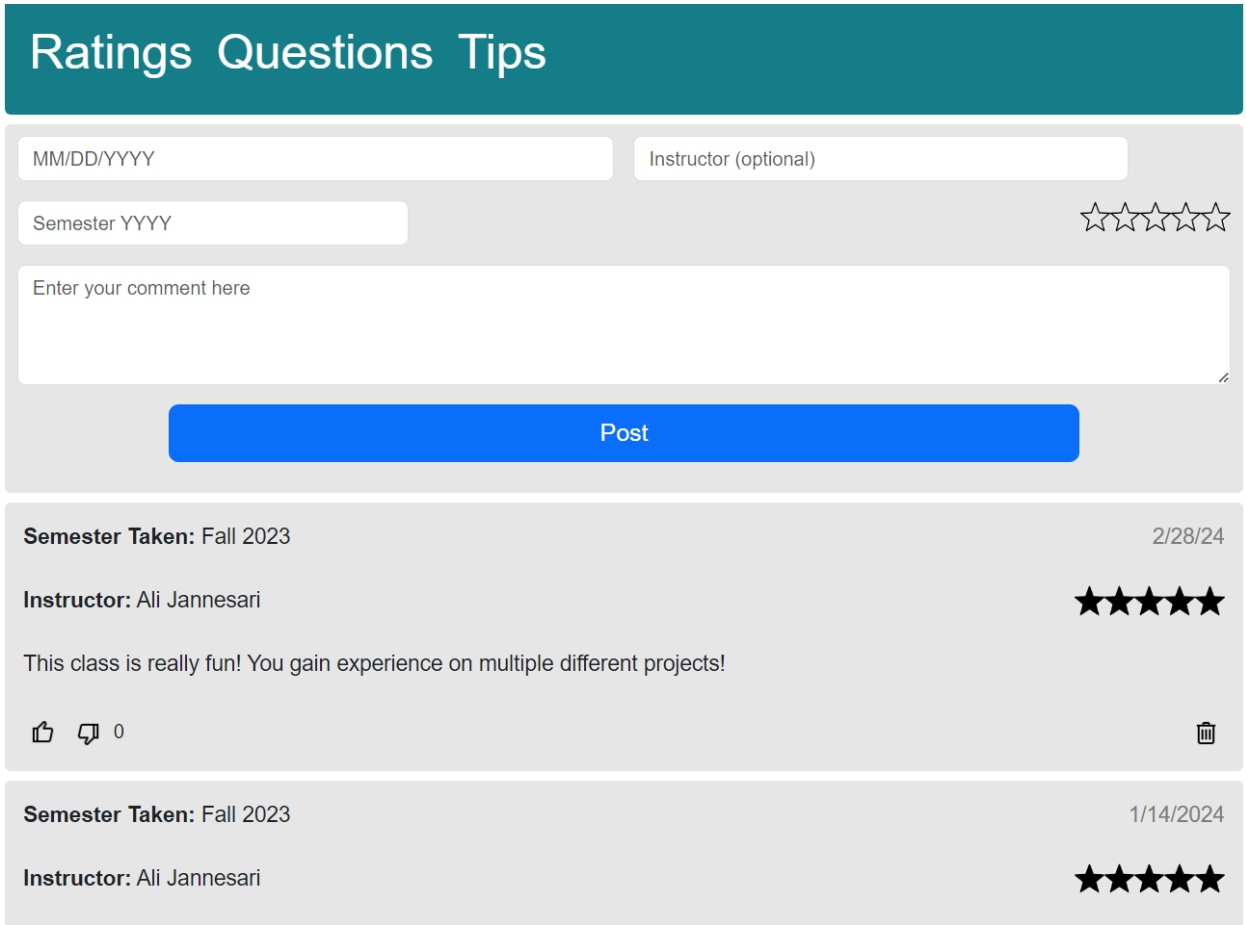
The view above is what a user will see upon initially getting to our website. This is also the view when a user clicks the ‘home’ button in the navigation bar. From this view a user can click on any class listed. For example, clicking on ‘COM S 319’ will take a user to a page with ratings for COM S 319.



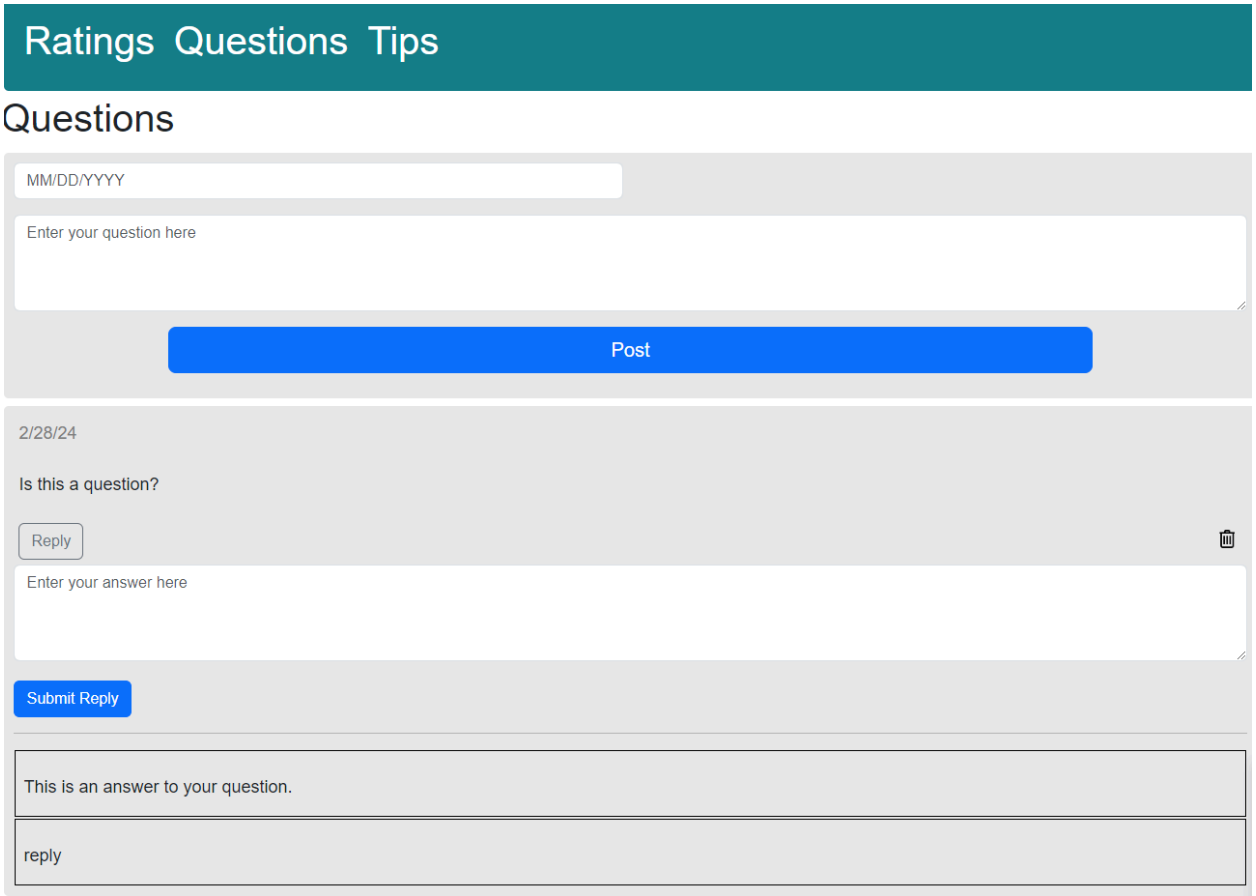
The view above displays the about page with information on the developers and contact information.



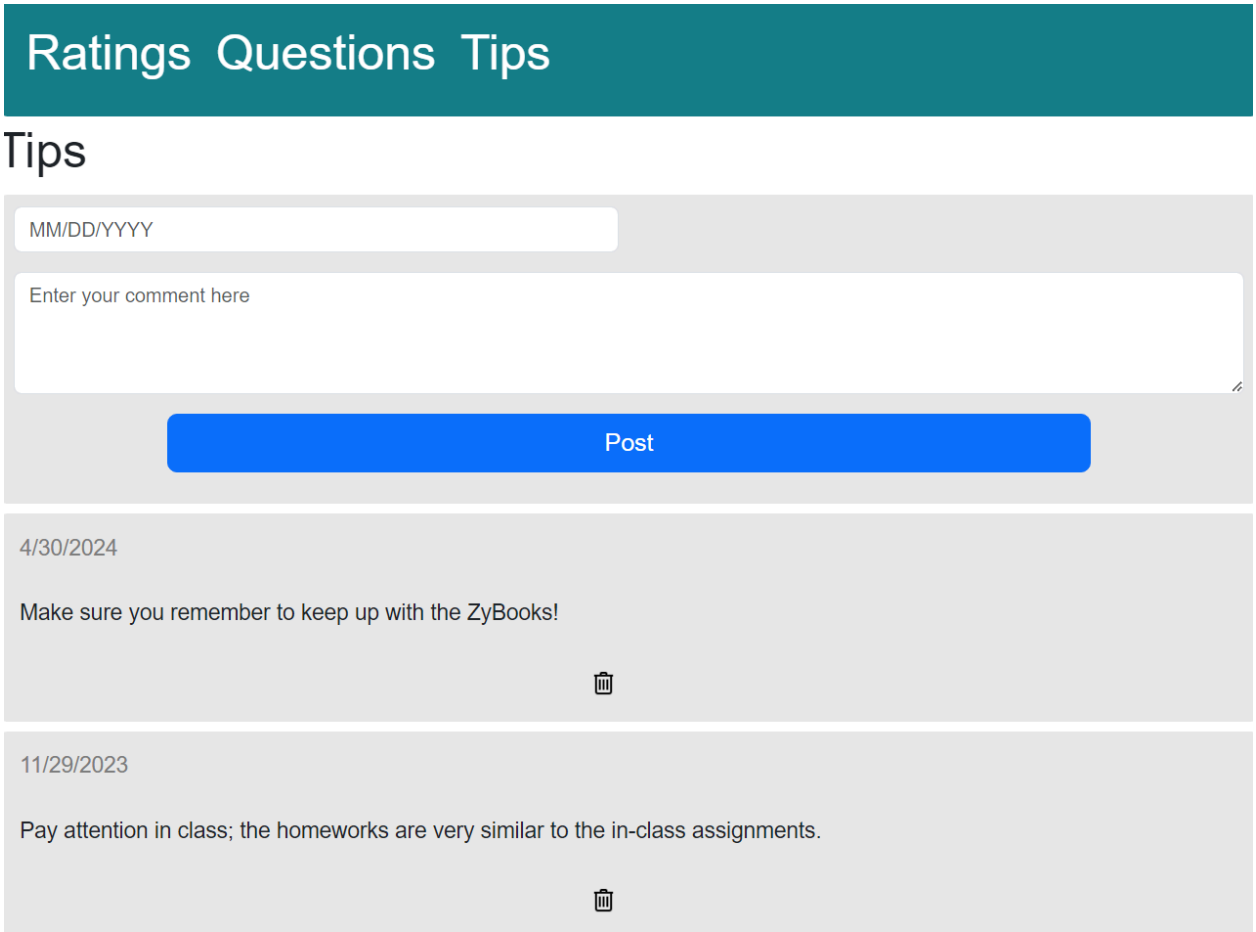
The view above displays only the information about the class the user selected, including the picture and description they saw on the home page. On this page the user can navigate between ‘Ratings’, ‘Questions’, and ‘Tips’ which are subpages.



The view above is a closer look at the ‘Ratings’ subpage. Here a user is able to add a review of the class. The inputs are: the date, instructor they took the class with (optional), the semester they took the class, their rating (stars), and the review itself. Below that is an example of a posted review. It displays all the inputs in addition to thumbs-up, thumbs-down, and trash can icons at the bottom. These icons allow the user to upvote, downvote, and delete posts, respectively.



The view above is a closer look at the ‘Questions’ subpage. A user is able to post a question, with the inputs being the date and their question. The question displays that along with the trash can icon, which lets a user delete a post, and a reply button. The reply button displays an input box like the one for posting a question and allows a user to reply to any question. Clicking the reply button will also hide the input box.



The view above is a closer look at the 'Tips' subpage. This subpage is very similar to the 'Questions' subpage with the same layout for users to input their comments. The displayed tips are also the same as the 'Questions' subpage, however, there is no reply button for this view.

Installation Manual

Disclaimer: these instructions assume you have downloaded Node.js and as well as React and all their dependencies beforehand.

- All of our files can be downloaded from the .zip folder submitted on Canvas or at https://github.com/mamcgovern/ComS319.git under 'team09_final' folder.
- After downloading all the files, move the frontend folder to another place temporarily for later. Then navigate to the 'team09_final' folder through a terminal / command prompt. This can be done by typing 'cd file_path' where file_path is the path to where you stored the downloaded files.
- Once you get to the correct folder (still in terminal/command prompt), type 'cd backend' to get to the backend folder.
- In the backend folder you'll want to install a couple things listed below (type the commands as written)
    - npm install -g nodemon
    - npm install express

    - npm install cors
    - npm install body-parser
    - npm install mongodb
    - npm init -y
- To run the backend type: node index.js
- Now navigate back to the team09_final folder
- Run the command: ‘npx create-react-app frontend’
- Then cd into the frontend folder and run the following commands:
    - npm install bootstrap
    - npm install react-hook-forms
- This next part can be done in the file explorer. Find the team09_final folder and go to the frontend folder you created earlier. Delete everything in the ‘src’ folder.
- Now (still in file explorer) find the previous frontend folder from the initial downloaded zip in the second step. Copy everything in the ‘src’ folder there and paste in the new ‘src’ folder where you just deleted everything.
- Back in the terminal / command prompt, navigate to the frontend folder.
- Run the command: ‘npm start’. This will start the React app.

MongoDB Setup:

- Our website uses MongoDB, so it’s necessary to download Mongo here.
- After that you’ll need to setup the database:
    - Make sure it’s connected at http://localhost:8081/
    - Create a database called “team09_final_db”
    - Create the following collections under that database called:
        - courses
        - ratings
        - questions
        - tips
- The data in each should look like the screenshots below:

  courses:

```
_id: ObjectId('663532cc1482d518cfbd750b')
id : 1
title : "Construction of User Interfaces"
courseCode : "COM S 319"
majorID : "COM S"
description : "Overview of user interface design. Evaluation and testing of user inte…"
imageURL : "https://img.freepik.com/free-photo/computer-program-coding-screen_5387…"
```

ratings:

```
_id: ObjectId('663532e21482d518cfbd7518')
id : 1
courseID : 1
date : "2/28/24"
semester : "Fall 2023"
professor : "Ali Jannesari"
stars : 5
helpful : NaN
unhelpful : 0
comment : "This class is really fun! You gain experience on multiple different pr…"
```

questions:

```
_id: ObjectId('663ade13cdcf246ce6c5b1f1')
id : 1
courseID : 1
date : "2/28/24"
question : "Is this a question?"
▾ answers : Array (2)
  ▾ 0: Object
      answer : "This is an answer to your question."
  ▾ 1: Object
      answer : "reply"
```

tips:

```
_id: ObjectId('66355a1b997de1ca7b665901')
id : 1
courseID : 1
date : "4/30/2024"
comment : "Make sure you remember to keep up with the ZyBooks!"
```

Code

/backend/index.js:

```
var express = require("express");
var cors = require("cors");
var app = express();
var fs = require("fs");
var bodyParser = require("body-parser");

app.use(cors());
app.use(bodyParser.json());
```

```
const port = "8081";
const host = "localhost";
const { MongoClient } = require("mongodb");

const url = "mongodb://127.0.0.1:27017";
const dbName = "team09_final_db";
const client = new MongoClient(url);
const db = client.db(dbName);

app.listen(port, () => {
    console.log("App listening at http://%s:%s", host, port);
});

// Get all courses
app.get("/courses/", async (req, res) => {
    await client.connect();
    console.log("Node connected successfully to GET MongoDB");
    const query = {};
    const results = await db
        .collection("courses")
        .find(query)
        .limit(100)
        .toArray();
    console.log(results);
    res.status(200);
    res.send(results);
});

// Get course by ID
app.get("/courses/:id", async (req, res) => {
    const courseid = Number(req.params.id);
    await client.connect();
    console.log("Node connected successfully to GET-id MongoDB");
    const query = { "id": courseid };
    const results = await db.collection("courses")
        .findOne(query);
    console.log("Results :", results);
    if (!results) res.send("Not Found").status(404);
    else res.send(results).status(200);
```

```
});

// Delete course by ID
app.delete("/courses/:id", async (req, res) => {
    try {
        const id = Number(req.params.id);
        await client.connect();
        console.log("Course to delete: ", id);
        const query = { id: id };
        // delete
        const results = await db.collection("courses").deleteOne(query);
        res.status(200);
        res.send(results);
    }
    catch (error) {
        console.error("Error deleting course:", error);
        res.status(500).send({ message: 'Internal Server Error' });
    }
});

// Post course
app.post("/courses/", async (req, res) => {
    try {
        await client.connect();
        const keys = Object.keys(req.body);
        const values = Object.values(req.body);

        const newDocument = {
            "id": values[0],
            "title": values[1],
            "courseCode": values[2],
            "majorID": values[3],
            "description": values[4],
            "imageURL": values[5]
        };
        console.log(newDocument);

        const results = await db
            .collection("courses")
            .insertOne(newDocument);
```

```
        res.status(200);
        res.send(results);
    } catch (error) {
        console.error("An error occurred:", error);
        res.status(500).send({ error: 'An internal server error occurred'
});
    }
});

// Get all ratings
app.get("/ratings", async (req, res) => {
    await client.connect();
    console.log("Node connected successfully to GET MongoDB");
    const query = {};
    const results = await db
        .collection("ratings")
        .find(query)
        .limit(100)
        .toArray();
    console.log(results);
    res.status(200);
    res.send(results);
});

// Get ratings by course ID
app.get("/ratings/:id", async (req, res) => {
    const courseid = Number(req.params.id);
    await client.connect();
    console.log("Node connected successfully to GET-id MongoDB");
    const query = { "courseID": courseid };
    const results = await db.collection("ratings")
        .find(query)
        .limit(100)
        .toArray();
    console.log("Results: ", results);
    if (!results) res.send("Not Found").status(404);
    else res.send(results).status(200);
});
```

```
// Put: Add Helpful
app.put("/ratings/helpful/:id", async (req, res) => {
    const id = Number(req.params.id);
    const query = { id: id };
    await client.connect();
    console.log("Rating to Update: ", id);
    // Data for updating the document, typically comes from the request
body
    console.log(req.body);
    const updateData = {
        $set: {
            "helpful": parseInt(req.body.helpful)
        }
    };
    // read data from rating to update to send to frontend
    const ratingUpdated = await db.collection("ratings").findOne(query);
    // Add options if needed, for example { upsert: true } to create a
document if it doesn't exist
    const options = {};
    const results = await db.collection("ratings").updateOne(query,
updateData, options);
    // If no document was found to update, you can choose to handle it by
sending a 404 response
    if (results.matchedCount === 0) {
        return res.status(404).send({ message: 'Rating not found' });
    }
    res.status(200);
    res.status(200).json({ results, updatedRating: ratingUpdated });
});

// Put: Add unhelpful
app.put("/ratings/unhelpful/:id", async (req, res) => {
    const id = Number(req.params.id);
    const query = { id: id };
    await client.connect();
    console.log("Rating to Update: ", id);
    // Data for updating the document, typically comes from the request
body
    console.log(req.body);
    const updateData = {
```

```
        $set: {
            "unhelpful": parseInt(req.body.unhelpful)
        }
    };
    // read data from rating to update to send to frontend
    const ratingUpdated = await db.collection("ratings").findOne(query);
    // Add options if needed, for example { upsert: true } to create a
document if it doesn't exist
    const options = {};
    const results = await db.collection("ratings").updateOne(query,
updateData, options);
    // If no document was found to update, you can choose to handle it by
sending a 404 response
    if (results.matchedCount === 0) {
        return res.status(404).send({ message: 'Rating not found' });
    }
    res.status(200);
    res.status(200).json({ results, updatedRating: ratingUpdated });
});


// Post rating
app.post("/ratings", async (req, res) => {
    try {
        await client.connect();
        const keys = Object.keys(req.body);
        const values = Object.values(req.body);

        const collection = db.collection('ratings');
        const maxIdDoc = await collection.findOne({}, { sort: { id: -1 }
});
        const maxId = maxIdDoc ? maxIdDoc.id : 0;

        const newId = maxId + 1;

        const newDocument = {
            "id": newId,
            "courseID": values[0],
            "date": values[1],
            "semester": values[2],
```

```
            "professor": values[3],
            "stars": parseInt(values[4]),
            "helpful": 0,
            "unhelpful": 0,
            "comment": values[5]
        };
        console.log(newDocument);


        const results = await db
            .collection("ratings")
            .insertOne(newDocument);
        res.status(200);
        res.send(results);
    } catch (error) {
        console.error("An error occurred:", error);
        res.status(500).send({ error: 'An internal server error occurred'
});
    }
});

// Delete rating by ID
app.delete("/ratings/:id", async (req, res) => {
    try {
        const id = Number(req.params.id);
        await client.connect();
        console.log("Rating to delete: ", id);
        const query = { id: id };
        // delete
        const results = await db.collection("ratings").deleteOne(query);
        res.status(200);
        res.send(results);
    }
    catch (error) {
        console.error("Error deleting rating:", error);
        res.status(500).send({ message: 'Internal Server Error' });
    }
});

/* Backend for Questions view */
```

```
// Get all questions
app.get("/questions", async (req, res) => {
    await client.connect();
    console.log("Node connected successfully to GET MongoDB");
    const query = {};
    const results = await db
        .collection("questions")
        .find(query)
        .limit(100)
        .toArray();
    console.log(results);
    res.status(200);
    res.send(results);
});

// Get questions by course ID
app.get("/questions/:id", async (req, res) => {
    const courseid = Number(req.params.id);
    await client.connect();
    console.log("Node connected successfully to GET-id MongoDB");
    const query = { "courseID": courseid };
    const results = await db.collection("questions")
        .find(query)
        .limit(100)
        .toArray();
    console.log("Results: ", results);
    if (!results) res.send("Not Found").status(404);
    else res.send(results).status(200);
});

// Post questions
app.post("/questions", async (req, res) => {
    try {
        await client.connect();
        const keys = Object.keys(req.body);
        const values = Object.values(req.body);

        const collection = db.collection('questions');
```

```
        const maxIdDoc = await collection.findOne({}, { sort: { id: -1 }
});
        const maxId = maxIdDoc ? maxIdDoc.id : 0;

        const newId = maxId + 1;

        const newDocument = {
            "id": newId,
            "courseID": values[0],
            "date": values[1],
            "question": values[2],
            "answers": values[3]
        };
        console.log(newDocument);


        const results = await db
            .collection("questions")
            .insertOne(newDocument);
        res.status(200);
        res.send(results);
    } catch (error) {
        console.error("An error occurred:", error);
        res.status(500).send({ error: 'An internal server error occurred'
});
    }
});

// Delete question by ID
app.delete("/questions/:id", async (req, res) => {
    try {
        const id = Number(req.params.id);
        await client.connect();
        console.log("question to delete: ", id);
        const query = { id: id };
        // delete
        const results = await db.collection("questions").deleteOne(query);
        res.status(200);
        res.send(results);
    }
```

```
    catch (error) {
        console.error("Error deleting question:", error);
        res.status(500).send({ message: 'Internal Server Error' });
    }
});

// Put: Add answer
app.put("/questions/:id", async (req, res) => {
    const id = Number(req.params.id);
    const query = { id: id };
    await client.connect();
    console.log("Rating to Update: ", id);
    // Data for updating the document, typically comes from the request
body
    console.log(req.body);
    const updateData = {
        $set: {
            "answers": req.body.answers
        }
    };
    // read data from question to update to send to frontend
    const questionUpdated = await
db.collection("questions").findOne(query);
    // Add options if needed, for example { upsert: true } to create a
document if it doesn't exist
    const options = {};
    const results = await db.collection("questions").updateOne(query,
updateData, options);
    // If no document was found to update, you can choose to handle it by
sending a 404 response
    if (results.matchedCount === 0) {
        return res.status(404).send({ message: 'Rating not found' });
    }
    res.status(200);
    res.status(200).json({ results, updatedRating: questionUpdated });
});

/* Backend for Tips view */

// Get all tips
```

```
app.get("/tips", async (req, res) => {
    await client.connect();
    console.log("Node connected successfully to GET MongoDB");
    const query = {};
    const results = await db
        .collection("tips")
        .find(query)
        .limit(100)
        .toArray();
    console.log(results);
    res.status(200);
    res.send(results);
});

// Get tips by course ID
app.get("/tips/:id", async (req, res) => {
    const courseid = Number(req.params.id);
    await client.connect();
    console.log("Node connected successfully to GET-id MongoDB");
    const query = { "courseID": courseid };
    const results = await db.collection("tips")
        .find(query)
        .limit(100)
        .toArray();
    console.log("Results: ", results);
    if (!results) res.send("Not Found").status(404);
    else res.send(results).status(200);
});

// Post tips
app.post("/tips", async (req, res) => {
    try {
        await client.connect();
        const keys = Object.keys(req.body);
        const values = Object.values(req.body);

        const collection = db.collection('tips');
        const maxIdDoc = await collection.findOne({}, { sort: { id: -1 }
});
        const maxId = maxIdDoc ? maxIdDoc.id : 0;
```

```
        const newId = maxId + 1;

        const newDocument = {
            "id": newId,
            "courseID": values[0],
            "date": values[1],
            "comment": values[2]
            // "helpful": 0,
            // "unhelpful": 0
        };
        console.log(newDocument);


        const results = await db
            .collection("tips")
            .insertOne(newDocument);
        res.status(200);
        res.send(results);
    } catch (error) {
        console.error("An error occurred:", error);
        res.status(500).send({ error: 'An internal server error occurred'
});
    }
});


//PUT tips

app.put("/tips/:id", async (req, res) => {
    const id = Number(req.params.id);
    const query = { id: id };
    await client.connect();
    console.log("Tip to Update: ", id);
    console.log(req.body);
    const updateData = {
        $set: {
            "tips": req.body.tips
        }
    };
```

```
    const tipUpdated = await db.collection("tips").findOne(query);
    const options = {};
    const results = await db.collection("tips").updateOne(query,
updateData, options);

    if (results.matchedCount === 0) {
        return res.status(404).send({ message: 'Tip not found' });
    }
    res.status(200).json({ results, updatedTip: tipUpdated });
});

// Delete tip by ID
app.delete("/tips/:id", async (req, res) => {
    try {
        const id = Number(req.params.id);
        await client.connect();
        console.log("Tip to delete: ", id);
        const query = { id: id };
        // delete
        const results = await db.collection("tips").deleteOne(query);
        res.status(200);
        res.send(results);
    }
    catch (error) {
        console.error("Error deleting tip:", error);
        res.status(500).send({ message: 'Internal Server Error' });
    }
});
```

/frontend/index.js and /frontend/Ratings.js:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import Ratings from './Ratings';

const root = ReactDOM.createRoot(document.getElementById('root'));
```

```
root.render(
  <React.StrictMode>
    <Ratings />
  </React.StrictMode>
);
```

```
import { useState, useEffect, useRef } from "react";
import { useForm } from "react-hook-form";
import "bootstrap/dist/css/bootstrap.css";
import starFilled from './images/star-fill.png';
import starOutline from './images/star-outline.png';
import thumbsUp from './images/thumbs-up.png'
import thumbsDown from './images/thumbs-down.png'
import trash from './images/trash.png';
import brokenRobot from './images/broken-robot.png';

function Ratings() {
    /*
     * 0: Courses
     * 1: One Course
     * 2: Students
     */
    const [view, setView] = useState(0);
    /*
     * 0: Ratings
     * 1: Questions
     * 2: Tips
     */
    const [courseView, setCourseView] = useState(0);

    const [allCourses, setAllCourses] = useState([]);
    const [courses, setCourses] = useState([]);
    const [course, setCourse] = useState([]);
    const [id, setInput] = useState();
    const [ratings, setRatings] = useState([]);
    const [questions, setQuestions] = useState([]);
    const [tips, setTips] = useState([]);
    const [rating, setRating] = useState(0);
    const [query, setQuery] = useState();
```

```
    const { register, handleSubmit, formState: { errors }, unregister,
reset } = useForm();
    const formRef = useRef(null);
    const [showReplyInput, setShowReplyInput] = useState(false);
    const [replyQuestionID, setReplyQuestionID] = useState(null);
    const [existingAnswers, setExistingAnswers] = useState([]);
    const { register: registerReply, handleSubmit: handleSubmitReply,
formState: { errors: replyErrors },  reset: replyReset} = useForm();
    /*
     * This method updates the courses array.
     */
    useEffect(() => {
        fetch("http://localhost:8081/courses/")
            .then(response => response.json())
            .then(courses => {
                setCourses(courses);
            })
    }, []);

    useEffect(() => {
        fetch("http://localhost:8081/courses/")
            .then(response => response.json())
            .then(courses => {
                setAllCourses(courses);
            })
    }, []);

    /*
     * This is the frontend method for getting one class by ID
     * (get request)
     */
    function getOneCourse(id) {
        setCourseView(0);
        setInput(id);
        fetch("http://localhost:8081/courses/" + id)
            .then(response => response.json())
            .then(course => { setCourse(course) });

        // get course ratings
        fetch("http://localhost:8081/ratings/" + id)
```

```
            .then(response => response.json())
            .then(ratings => { setRatings(ratings) });
        setView(1);

        // get course questions
        fetch("http://localhost:8081/questions/" + id)
            .then(response => response.json())
            .then(questions => { setQuestions(questions) });
        // setView(1);

        // get course tips
        fetch("http://localhost:8081/tips/" + id)
            .then(response => response.json())
            .then(tips => { setTips(tips) });
        // setView(1);
    }

    /*
     * This is the frontend method for liking a rating
     * (put request)
     */
    function addHelpful(ratingID, helpfulCount) {
        console.log(ratingID);
        fetch(`http://localhost:8081/ratings/helpful/${ratingID}`, {
            method: 'PUT',
            headers: { 'content-type': 'application/json' },
            body: JSON.stringify(
                {
                    "helpful": helpfulCount
                }
            )
        })
            .then(response => response.json())
            .then(() => {
                // Fetch updated ratings after updating unhelpful count
                fetch(`http://localhost:8081/ratings/${id}`)
                    .then(response => response.json())
                    .then(ratings => {
                        setRatings(ratings);
                    });
```

```
            });
    }

    /*
     * This is the frontend method for disliking a rating
     * (put request)
     */
    function addUnhelpful(ratingID, unhelpfulCount) {
        console.log(ratingID);
        fetch(`http://localhost:8081/ratings/unhelpful/${ratingID}`, {
            method: 'PUT',
            headers: { 'content-type': 'application/json' },
            body: JSON.stringify(
                {
                    "unhelpful": unhelpfulCount
                }
            )
        })
            .then(response => response.json())
            .then(() => {
                fetch(`http://localhost:8081/ratings/${id}`)
                    .then(response => response.json())
                    .then(ratings => {
                        setRatings(ratings);
                    });
            });
    }


    /*
     * This function deletes a rating by ID
     */
    function deleteRating(ratingID) {
        const confirmed = window.confirm("Are you sure you want to delete
this rating: " + ratingID);
        if (confirmed) {
            console.log('Confirmed');
            fetch(`http://localhost:8081/ratings/${ratingID}`, {
                method: 'DELETE',
                headers: { 'content-type': 'application/json' },
```

```
                body: JSON.stringify(
                    { "id": ratingID }
                )
            })
                .then(response => response.json())
                .then(() => {
                    fetch(`http://localhost:8081/ratings/${id}`)
                        .then(response => response.json())
                        .then(ratings => {
                            setRatings(ratings);
                        });
                });
        } else {
            console.log('Canceled');
        }
    }

    /*
     * This function updates the number of stars
     */
    const handleRatingChange = (newRating) => {
        setRating(newRating); // Update the rating state with the new
rating
    };

    /*
     * This function is for submitting the rating info.
     * It includes a post request.
     */
    const onSubmit = (data, event) => {
        event.preventDefault(); // Prevent default form submission
behavior
        console.log(data); // log all data

        const ratingData = {
            courseID: id,
            date: data.date,
            semester: data.semester,
            professor: data.professor,
            stars: rating,
```

```
            comment: data.comment
        };

        fetch('http://localhost:8081/ratings', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(ratingData)
        })
            .then(response => response.json())
            .then(responseData => {
                console.log('Success:', responseData);
            })
            .then(() => {
                // Fetch updated ratings after posting a new rating
                fetch(`http://localhost:8081/ratings/${id}`)
                    .then(response => response.json())
                    .then(ratings => {
                        setRatings(ratings);
                    });
            })
            .catch(error => console.error('Error fetching product data:',
error));
        reset();
    };

    /*
    * This function deletes a question by ID
    */
    function deleteQuestion(questionID) {
        const confirmed = window.confirm("Are you sure you want to delete
this question: " + questionID);
        if (confirmed) {
            console.log('Confirmed');
            fetch(`http://localhost:8081/questions/${questionID}`, {
                method: 'DELETE',
                headers: { 'content-type': 'application/json' },
                body: JSON.stringify(
                    { "id": questionID }
```

```
                )
            })
                .then(response => response.json())
                .then(() => {
                    fetch(`http://localhost:8081/questions/${id}`)
                        .then(response => response.json())
                        .then(questions => {
                            setQuestions(questions);
                        });
                });
        } else {
            console.log('Canceled');
        }
    }

    /**
     *
     * This function is for submitting questions
     */
    const submitQuestions = (data, event) => {
        event.preventDefault(); // Prevent default form submission
behavior
        console.log(data); // log all data

        const questionData = {
            courseID: id,
            date: data.date,
            question: data.question,
            answers: []
        };

        fetch('http://localhost:8081/questions', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(questionData)
        })
            .then(response => response.json())
            .then(responseData => {
```

```
                console.log('Success:', responseData);
            })
            .then(() => {
                // Fetch updated ratings after posting a new rating
                fetch(`http://localhost:8081/questions/${id}`)
                    .then(response => response.json())
                    .then(questions => {
                        setQuestions(questions);
                    });
            })
            .catch(error => console.error('Error fetching product data:',
error));

        reset();
    };

    /*
     * This handles replies/answers to questions
     */
    const submitReply = (data, event) => {
        event.preventDefault(); // Prevent default form submission
behavior
        addAnswer(replyQuestionID, data.answer);
        setShowReplyInput(false);
        // Reset the form fields after a short delay
        replyReset();
    };

    /*
     * This is the frontend method for adding an answer to a question
     * (put request)
     */
    function addAnswer(questionID, answerString) {
        // Combine existing answers with the new answer
        const updatedAnswers = [...existingAnswers, { answer: answerString
}];

        fetch(`http://localhost:8081/questions/${questionID}`, {
            method: 'PUT',
            headers: { 'content-type': 'application/json' },
```

```
            body: JSON.stringify({
                answers: updatedAnswers
            })
        })
            .then(response => response.json())
            .then(() => {
                // After updating, fetch the updated question data
                fetch(`http://localhost:8081/questions/${questionID}`)
                    .then(response => response.json())
                    .then(questions => {
                        setRatings(questions);
                    });
            })
            .then(() => {
                fetch("http://localhost:8081/questions/" + id)
                    .then(response => response.json())
                    .then(questions => { setQuestions(questions) });
            });
    }

    /*
     * This function deletes a tip by ID
     */
    function deleteTips(tipID) {
        const confirmed = window.confirm("Are you sure you want to delete
this question: " + tipID);
        if (confirmed) {
            console.log('Confirmed');
            fetch(`http://localhost:8081/tips/${tipID}`, {
                method: 'DELETE',
                headers: { 'content-type': 'application/json' },
                body: JSON.stringify(
                    { "id": tipID }
                )
            })
                .then(response => response.json())
                .then(() => {
                    fetch(`http://localhost:8081/tips/${id}`)
                        .then(response => response.json())
                        .then(tips => {
```

```
                            setTips(tips);
                        });
                });
        } else {
            console.log('Canceled');
        }
    }

    /**
     *
     * This function is for submitting tips
     */
    const submitTips = (data, event) => {
        event.preventDefault(); // Prevent default form submission
behavior
        console.log(data); // log all data

        const tipsData = {
            courseID: id,
            date: data.date,
            comment: data.comment
        };

        fetch('http://localhost:8081/tips', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(tipsData)
        })
            .then(response => response.json())
            .then(responseData => {
                console.log('Success:', responseData);
            })
            .then(() => {
                // Fetch updated ratings after posting a new rating
                fetch(`http://localhost:8081/tips/${id}`)
                    .then(response => response.json())
                    .then(tips => {
                        setTips(tips);
```

```
                    });
            })
            .catch(error => console.error('Error fetching product data:',
error));

        reset();
    };


    const handleChange = (e) => {
        setQuery(e.target.value);
        const results = allCourses.filter(eachCourse => {
            if (e.target.value == "") return allCourses;
            return
eachCourse.courseCode.toLowerCase().includes(e.target.value.toLowerCase())
        });
        setCourses(results);
    }

    /*
     * This creates the navbar for each view so it doesn't have to be
retyped.
     */
    function navbar() {
        return (
            <div>
                <header data-bs-theme="dark">
                    <nav class="navbar navbar-expand-lg bg-body-tertiary
rounded" aria-label="Navbar">
                        <div class="container-fluid">
                            <div class="collapse navbar-collapse
d-lg-flex" id="navbarsExample11">
                                <button class="button-like-text
navbar-brand col-lg-3 me-0" onClick={() =>
setView(0)}>CourseRater</button>
                                <ul class="navbar-nav col-lg-6
justify-content-lg-center">
                                    <li class="nav-item" style={{ margin:
'5px' }}>
```

```
                                        <button class="btn
btn-outline-light btn-lg" onClick={() => setView(0)}>Home</button>
                                    </li>
                                    <li class="nav-item" style={{ margin:
'5px' }}>
                                        <button class="btn
btn-outline-light btn-lg" onClick={() => setView(2)}>About</button>
                                    </li>
                                </ul>
                                <input type="text" placeholder="Enter
Course ID" value={query} onChange={handleChange} />

                            </div>
                        </div>
                    </nav>
                </header>
            </div>
        )
    }

    /*
     * This view functions as the homepage.
     * It lists all of the courses in the database.
     */
    function viewCourses() {
        const allCourses = courses.map((el) => (
            <div class="subforum-row">
                <div class="subforum-column pic-div center">
                    <img src={el.imageURL} alt="..." style={{ width:
'100%' }} />
                </div>
                <div class="subforum-description subforum-column">
                    <h1>{el.id}: <button class="button-like-text"
type="button" onClick={() =>
getOneCourse(el.id)}>{el.courseCode}</button></h1>
                    <p class="title"><strong>{el.title}</strong></p>
                    <p>{el.description}</p>
                </div>
            </div>
        ));
```

```
        return (
            <div>
                {navbar()}
                <div class="container subforum">
                    <div class="subforum-title">
                        <h1>ISU Classes</h1>
                    </div>
                    <div>
                        {allCourses}
                    </div>
                </div>
            </div>
        )
    }

    /*
     * This function allows us to view one course.
     * it uses viewRatings, viewTips, and viewQuestions
     * to finish filling out the page.
     */
    function viewOneCourse() {
        function subsection() {
            if (courseView === 0) {
                return viewRatings();
            } else if (courseView === 1) {
                return viewQuestions();
            } else if (courseView === 2) {
                return viewTips();
            }
        }
        return (
            <div>
                {navbar()}
                <div class="subforum-container">
                    <div class="subforum" id="subforum">
                        <div class="subforum-title" id="classTitle">
                            <h1>{course.courseCode}</h1>
                        </div>
                        <div class="subforum-full-row"
id="classDescription">
```

```
                            <div class="subforum-description
subforum-column">
                                <img src={course.imageURL} alt="..."
style={{ width: '200px' }} />
                                <h1>{course.title}</h1>
                                <p>{course.description}</p>
                            </div>
                        </div>
                        <div class="subforum-subtitle">
                            <button type="button" id="ratings-link"
onClick={() => setCourseView(0)}><h1>Ratings</h1></button>
                            <button type="button" id="questions-link"
onClick={() => setCourseView(1)}><h1>Questions</h1></button>
                            <button type="button" id="tips-link"
onClick={() => setCourseView(2)}><h1>Tips</h1></button>
                        </div>
                        {subsection()}
                    </div>
                </div>

            </div>
        )
    }

    /*
     * This subview shows the ratings for a selected course.
     */
    function viewRatings() {
        function stars(numStars) {
            const stars = [];
            for (let i = 1; i <= 5; i++) {
                if (i <= numStars) {
                    stars.push(<img key={i} src={starFilled} style={{
width: '25px', cursor: 'pointer' }} alt="star filled" />);
                } else {
                    stars.push(<img key={i} src={starOutline} style={{
width: '25px', cursor: 'pointer' }} alt="star outline" />);
                }
            }
            return stars;
```

```
        }
        const renderStars = () => {
            const stars = [];
            for (let i = 1; i <= 5; i++) {
                if (i <= rating) {
                    stars.push(<img key={i} src={starFilled} style={{
width: '25px', cursor: 'pointer' }} alt="star filled" onClick={() =>
handleRatingChange(i)} />);
                } else {
                    stars.push(<img key={i} src={starOutline} style={{
width: '25px', cursor: 'pointer' }} alt="star outline" onClick={() =>
handleRatingChange(i)} />);
                }
            }
            return stars;
        };
        const allRatings = ratings.map((el) => (
            <div id="ratings" class="subforum-full-row">
                <div class="subforum-description subforum-column">
                    <div class="ratings-row">
                        <div class="ratings-column">
                            <p><strong>Semester Taken:</strong>
{el.semester} </p>
                        </div>
                        <div class="ratings-column" style={{ textAlign:
'right' }}>
                            <p class="date">{el.date}</p>
                        </div>
                    </div>
                    <div class="ratings-row">
                        <div class="ratings-column">
                            <p><strong>Instructor:</strong>
{el.professor}</p>
                        </div>
                        <div class="ratings-column" style={{ textAlign:
'right' }}>
                            {stars(el.stars)}
                        </div>
                    </div>
                    <div class="ratings-full-row">
```

```
                        <div class="ratings-column">
                            <p>{el.comment}</p>
                        </div>
                    </div>
                    <div class="ratings-row">
                        <div class="ratings-column">
                            <button class="button-like-text" onClick={()
=> addHelpful(el.id, el.helpful + 1)}><img src={thumbsUp} style={{ width:
'25px' }} alt="thumbs up" /> </button> {el.helpful}
                            <button class="button-like-text" onClick={()
=> addUnhelpful(el.id, el.unhelpful + 1)}><img src={thumbsDown} style={{
width: '25px' }} alt="thumbs down" /></button> {el.unhelpful}
                        </div>
                        <div class="ratings-column" style={{ textAlign:
'right' }}>
                            <button class="button-like-text" onClick={()
=> deleteRating(el.id)}><img src={trash} style={{ width: '25px' }}
alt="trash" /></button>
                        </div>
                    </div>
                </div>
            </div>
        ));

        return (
            <div>
                <div><h1>Ratings</h1></div>
                <div class="subforum-full-row">
                    <div class="subforum-column">
                        <div class="row g-3">
                            <div class="col">
                                <form class="needs-validation"
ref={formRef} onSubmit={handleSubmit(onSubmit)}>

                                    <div class="row g-3 mb-3">
                                        <div class="col-sm-6">
                                            <div className="form-group">
                                                <input
{...register("date", { required: true, pattern:
```

```
/^(0[1-9]|1[0-2])\/(0[1-9]|[12]\d|3[01])\/\d{4}$/ })}
placeholder="MM/DD/YYYY" className="form-control" />
                                                    {errors.date && <p
className="text-danger">Date must be in form MM/DD/YYYY.</p>}
                                                </div>
                                            </div>
                                            <div class="col-md-5">
                                                <div className="form-group">
                                                    <input
{...register("professor")} placeholder="Instructor (optional)"
className="form-control" />
                                                </div>
                                            </div>
                                        </div>
                                        <div class="row g-3 mb-3">


                                            <div class="col-md-4">
                                                <div class="form-group">
                                                    <input
{...register("semester", { required: true, pattern:
/^(Spring|Summer|Fall|Winter)\s\d{4}$/ })} placeholder="Semester YYYY"
className="form-control" />
                                                    {errors.semester && <p
className="text-danger">Semester is required.</p>}
                                                </div>
                                            </div>

                                            <div class="col-md-5">
                                                <div class="form-group">
                                                </div>
                                            </div>

                                            <div class="col-md-3 ">
                                                <div class="form-group"
style={{ textAlign: 'right' }}>
                                                    {renderStars()}
                                                </div>
                                            </div>
                                        </div>
```

```
                                    <div class="row g-3 mb-3">
                                        <div class="col">
                                            <textarea
                                                {...register("comment", {
required: true })}
                                                placeholder="Enter your
comment here"
                                                className="form-control"
                                                style={{ width: '100%',
minHeight: '100px', resize: 'both' }}
                                            />
                                            {errors.comment && <p
className="text-danger">Comment is required.</p>}
                                        </div>
                                    </div>

                                    {/* Submit Button */}
                                    <div class="row g-3 mb-3">
                                        <div class="col" style={{
textAlign: 'center' }}>
                                            <button class=" btn
btn-primary btn-lg" type="submit" style={{ width: '75%' }}>Post</button>
                                        </div>
                                    </div>
                                </form>
                            </div>
                        </div>
                    </div>
                </div>
                {allRatings}
            </div>
        )
    }

    /*
     * This subview shows the questions for a selected course.
     */
    function viewQuestions() {
        // Function to handle showing/hiding reply input area
        const toggleReplyInput = (questionID, ans) => {
```

```
            setShowReplyInput(!showReplyInput);
            setReplyQuestionID(questionID);
            setExistingAnswers(ans);
        };

        // Render reply input area if showReplyInput is true
        const renderReplyInput = (questionID) => {
            if (showReplyInput && replyQuestionID === questionID) {
                return (
                    <div>
                        <form onSubmit={handleSubmitReply(submitReply)}>
                            {/* Form inputs for reply */}
                            <textarea {...registerReply("answer", {
required: true })}
                                placeholder="Enter your answer here"
                                className="form-control"
                                style={{ width: '100%', minHeight:
'100px', resize: 'both' }} />
                            {replyErrors.answer && <p
className="text-danger">This field is required</p>}
                            <button class=" btn btn-primary" type="submit"
style={{ marginTop: '20px' }}>Submit Reply</button>
                        </form>
                    </div>
                );
            }
            return null;
        };

        const allQuestions = questions.map((el) => (
            <div id="ratings" className="subforum-full-row">
                <div className="subforum-description subforum-column">
                    <div className="ratings-row">
                        <div className="ratings-column">
                            <p className="date">{el.date}</p>
                        </div>
                    </div>
                    <div class="ratings-full-row">
                        <div class="ratings-column">
                            <p>{el.question}</p>
```

```
                        </div>
                    </div>
                    <div className="ratings-row">
                        <div className="ratings-column">
                            <button type="button" className="btn
btn-outline-secondary" onClick={() => toggleReplyInput(el.id,
el.answers)}>Reply</button>
                        </div>
                        <div className="ratings-column" style={{
textAlign: 'right' }}>
                            <button className="button-like-text"
onClick={() => deleteQuestion(el.id)}><img src={trash} style={{ width:
'25px' }} alt="trash" /></button>
                        </div>
                    </div>
                    <div>{renderReplyInput(el.id)}</div>
                    <div><hr></hr></div>
                    <div className="ratings-full-row">
                        {el.answers.map((e) => (
                            <div style={{ margin: '1px', border: '1px
solid black', paddingLeft: '8px' }}>
                                <br />
                                <p>{e.answer}</p>
                            </div>
                        ))}
                    </div>
                </div>
            </div>
        ));

        return (
            <div>
                <div><h1>Questions</h1></div>
                <div class="subforum-full-row">
                    <div class="subforum-column">
                        <div class="row g-3">
                            <div class="col">
                                <form class="needs-validation"
ref={formRef} onSubmit={handleSubmit(submitQuestions)}>
```

```
                                    <div class="row g-3 mb-3">
                                        <div class="col-sm-6">
                                            <div className="form-group">
                                                <input
{...register("date", { required: true, pattern:
/^(0[1-9]|1[0-2])\/(0[1-9]|[12]\d|3[01])\/\d{4}$/ })}
placeholder="MM/DD/YYYY" className="form-control" />
                                                {errors.date && <p
className="text-danger">Date must be in form MM/DD/YYYY.</p>}
                                            </div>
                                        </div>
                                    </div>

                                    <div class="row g-3 mb-3">
                                        <div class="col">
                                            <textarea
                                                {...register("question", {
required: true })}
                                                placeholder="Enter your
question here"
                                                className="form-control"
                                                style={{ width: '100%',
minHeight: '100px', resize: 'both' }}
                                            />
                                            {errors.question && <p
className="text-danger">Question is required.</p>}
                                        </div>
                                    </div>

                                    {/* Submit Button */}
                                    <div class="row g-3 mb-3">
                                        <div class="col" style={{
textAlign: 'center' }}>
                                            <button class=" btn
btn-primary btn-lg" type="submit" style={{ width: '75%' }}>Post</button>
                                        </div>
                                    </div>
                                </form>
                            </div>
                        </div>
```

```
                    </div>
                </div>
                {allQuestions}
            </div>
        );
    }



    /*
     * This subview shows the tips for a selected course.
     */
    function viewTips() {
        const allTips = tips.map((el) => (
            <div id="ratings" class="subforum-full-row">
                <div class="subforum-description subforum-column">
                    <div class="ratings-row">
                        <div class="ratings-column" >
                            <p class="date">{el.date}</p>
                        </div>
                    </div>
                    <div class="ratings-full-row">
                        <div class="ratings-column">
                            <p>{el.comment}</p>
                        </div>
                    </div>
                    <div class="ratings-row">
                        {/* <div class="ratings-column">
                            <button class="button-like-text" onClick={()
=> addHelpful(el.id, el.helpful + 1)}><img src={thumbsUp} style={{ width:
'25px' }} alt="thumbs up" /> </button> {el.helpful}
                            <button class="button-like-text" onClick={()
=> addUnhelpful(el.id, el.unhelpful + 1)}><img src={thumbsDown} style={{
width: '25px' }} alt="thumbs down" /></button> {el.unhelpful}
                        </div> */}
                        <div class="ratings-column" style={{ textAlign:
'right' }}>
                            <button class="button-like-text" onClick={()
=> deleteTips(el.id)}><img src={trash} style={{ width: '25px' }}
alt="trash" /></button>
                        </div>
```

```
                    </div>
                </div>
            </div>
        ));

        return (
            <div>
                <div><h1>Tips</h1></div>
                <div class="subforum-full-row">
                    <div class="subforum-column">
                        <div class="row g-3">
                            <div class="col">
                                <form class="needs-validation"
ref={formRef} onSubmit={handleSubmit(submitTips)}>

                                    <div class="row g-3 mb-3">
                                        <div class="col-sm-6">
                                            <div className="form-group">
                                                <input
{...register("date", { required: true, pattern:
/^(0[1-9]|1[0-2])\/(0[1-9]|[12]\d|3[01])\/\d{4}$/ })}
placeholder="MM/DD/YYYY" className="form-control" />
                                                {errors.date && <p
className="text-danger">Date must be in form MM/DD/YYYY.</p>}
                                            </div>
                                        </div>
                                    </div>

                                    <div class="row g-3 mb-3">
                                        <div class="col">
                                            <textarea
                                                {...register("comment", {
required: true })}
                                                placeholder="Enter your
comment here"
                                                className="form-control"
                                                style={{ width: '100%',
minHeight: '100px', resize: 'both' }}
                                            />
```

```
                                                {errors.comment && <p
className="text-danger">Comment is required.</p>}
                                            </div>
                                        </div>

                                        {/* Submit Button */}
                                        <div class="row g-3 mb-3">
                                            <div class="col" style={{
textAlign: 'center' }}>
                                                <button class=" btn
btn-primary btn-lg" type="submit" style={{ width: '75%' }}>Post</button>
                                            </div>
                                        </div>
                                    </form>
                                </div>
                            </div>
                        </div>
                    </div>
                    {allTips}
                </div>
            )
        }

        /*
         * This view shows the about page,
         * including information about the course, students, and assignment.
         */
        function viewStudents() {
            return (
                <div>
                    {/* Header */}
                    {navbar()}
                    <div class="container">
                        <div class="container subforum">
                            <div class="subforum-title">
                                <h1>About</h1>
                            </div>
                            <div>
                                <div class="subforum-full-row">
                                    <div class="subforum-column">
```

```
                                    <h2>Class Information</h2>
                                    <p>Com S 319: Construction of User
Interfaces</p>
                                    <p>Professor Ali Jannesari</p>
                                    <p>May 9, 2024</p>
                                </div>
                            </div>
                            <div class="subforum-full-row">
                                <div class="subforum-column">
                                    <h2>Project Description</h2>
                                    <p class="title">Final Project</p>
                                    <p>This website is a ratings page that
allows students to leave reviews
                                        of the classes they've taken. The
website was developed using MERN (MongoDB, Express, React, NodeJS)
                                        along with bootstrap. Students are
able to read reviews of various classes at ISU and leave their own
                                        reviews, as well as questions or
tips about the class.
                                    </p>
                                </div>
                            </div>
                            <div class="subforum-row-thirds">
                                <div class="subforum-column">
                                    <h3>Maddelynne McGovern</h3>
                                    <p class="title">mrm4@iastate.edu</p>
                                </div>
                                <div class="subforum-column">
                                    <h3>Jennifer Hua</h3>
                                    <p class="title">jthua@iastate.edu</p>
                                </div>
                                <div class="subforum-column">
                                    <h3>Sagnik Dey</h3>
                                    <p class="title">sdey@iastate.edu</p>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
```

```
        );
    }

    /*
     * 1: Courses
     * 2: Ratings
     * 3: Questions
     * 4: Tips
     * 5: Students
     */

    /*
     * This if/else statement sets the view.
     */
    if (view === 0) {
        return viewCourses();
    } else if (view === 1) {
        return viewOneCourse();
    } else if (view === 2) {
        return viewStudents();
    } else {
        return (
            <div>
                {navbar()}
                <div class="subforum-container">
                    <div class="subforum" id="subforum">
                        <div class="subforum-title" id="classTitle">
                            <h1>Uh oh...</h1>
                        </div>
                        <div class="subforum-full-row">
                            <div class="subforum-column" style={{
textAlign: 'center' }}>
                                <img src={brokenRobot} alt="Broken Robot"
style={{ width: '100px' }} />
                                <p class="title">Error 404: Page Not
Found</p>
                                <p>The page you are looking for may have
been moved, deleted, or possibly never existed.</p>
```

```
                                <p>Click <button class="button-like-text"
style={{ color: '#177e89' }} onClick={() => setView(0)}>here</button> to
go back home.</p>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        );
    }
}

export default Ratings;
```