Milestone 4 (Team C)

Product summary

MentorMe is a web-based platform with the goal of revolutionizing the tutoring industry. Its goal is to link students with the top teachers and SMEs in the field.

List of final product functions:

- New students should be able to register using their university email addresses.
- Students must log in to the system using their university email.
- Students should have access to the system and be able to search for tutors.
- A message request to the tutor can be sent by students.
- Students must be able to have real-time messages with the tutor once the tutor has accepted their request.
- A tutor's course must be available for students to evaluate.
- The tutor must be able to register in the system and use it without any problem once the verification is completed by the Moderator.
- The tutor must be able to log in to the system.
- The tutor shall be able to upload his/her CV & image.
- The tutor must be able to accept a student's communication request.
- When new Tutor information arrives, the system must tell the Moderator.
- The moderator should be able to approve/reject content posted by the tutor before making it live (CV, Image).
- The tutor and the Students must be able to update their profiles.

What's Unique??

Ratings
Ban of a Student Profile or of a tutor profile by a moderator.
Chat Request Functionality

URL of the deployed application: http://20.113.70.224/

Usability Test Plan

Test objectives

On our website MentorMe, we have decided to test the functionality of Search along with the filters that we have on the website.

Our main concern is to check that if a naive/new user tries to use our application for the first time, how easily/comfortably he/she can use the application without any problems. We want to make sure that the design of the application is user-friendly in every possible way so that they don't need any kind of manual to use our application, even for the very first time. We check for things like: "Will users easily find the search box in its present location?" or "Will the user be able to find the filter functionality and use it easily along with the search option that he has."

Test background and setup

System setup: To perform this test Internet connection and a device with a browser.

Starting point: In order to conduct usability testing the starting point for a user is to open a browser and enter our website's URL.

Who are the intended users: We have selected our users and they can be any student of HS Fulda who is not part of our project, directly or indirectly.

URL of the system to be tested: http://20.113.70.224/searchTutors

What is to be measured: We have to measure the flexibility of our website, we plan to work further if we find some of the things in the Usability testing is not so easy or comfortable for our users to find/navigate through.

So, for user Satisfaction evaluation we will be using the Likert Test which includes assessing opinions, attitudes, or behaviors of our users.

Usability Task description

In order to perform the test, the tester first need to navigate to the URL provided above and then they should try to search something by putting text into the search box then they should observe that did they find anything relevant or not and to carry forward this test further they can also sort this result according to price or rating. Apart from this testing tester should also check this page by loading in various different sizes of devices like mobile, tablet, laptop to check how UI is getting rendered on different devices. Finally, they should evaluate the result.
To measure effectiveness, first, the tester should check, are results actually according to what they wrote in the search box and if the tester has selected any sorting option then, is the result sorted according to the given criteria?

Questionnaire

Question - 1. How satisfied were you with the UI and Functionality of this page?

1. Very dissatisfied
2. Somewhat dissatisfied.
3. Neither satisfied nor dissatisfied.
4. Somewhat satisfied.
5. Very satisfied.

Question - 2. How well does this page meet your needs?

1. It did not meet my needs at all.
2. It met very few of my needs.
3. It met some of my needs.
4. It met the majority of my needs.
5. It met all of my needs.

Question - 3. How intuitive did you find using Search Box and Sorting Options?

1. Not intuitive at all.
2. Not very intuitive.
3. Somewhat intuitive.
4. Mostly intuitive.
5. Extremely intuitive.

Tester: Aurangazeb Khan (Fellow classmate who is not part of GDSD Project this semester)

Test Results:

Question - 1. How satisfied were you with the UI and Functionality of this page?

Very dissatisfied	
Somewhat dissatisfied.	
Neither satisfied nor dissatisfied.	
Somewhat satisfied.	
Very satisfied.	**X**

Question - 2. How well does this page meet your needs?

It did not meet my needs at all.	
It met very few of my needs.	
It met some of my needs.	
It met the majority of my needs.	**X**
It met all of my needs.	

Question - 3. How intuitive did you find using Search Box and Sorting Options?

Not intuitive at all.	
Not very intuitive.	
Somewhat intuitive.	
Mostly intuitive.	
Extremely intuitive.	**X**

Tester: Talha Jahangiri Khan (Fellow classmate who is part of GDSD Project this semester)

Test Results:

Question - 1. How satisfied were you with the UI and Functionality of this page?

Very dissatisfied	
Somewhat dissatisfied.	
Neither satisfied nor dissatisfied.	
Somewhat satisfied.	**X**
Very satisfied.	

Question - 2. How well does this page meet your needs?

It did not meet my needs at all.	

It met very few of my needs.

It met some of my needs.

It met the majority of my needs.

It met all of my needs. **X**

Question - 3. How intuitive did you find using Search Box and Sorting Options?

Not intuitive at all.

Not very intuitive.

Somewhat intuitive.

Mostly intuitive. **X**

Extremely intuitive.

Tester: Hamza Khalid (Fellow student of HS Fulda)

Test Results:

Question - 1. How satisfied were you with the UI and Functionality of this page?

Very dissatisfied

Somewhat dissatisfied.

Neither satisfied nor dissatisfied.

Somewhat satisfied.

Very satisfied. **X**

Question - 2. How well does this page meet your needs?

It did not meet my needs at all.

It met very few of my needs.

It met some of my needs.

It met the majority of my needs.

It met all of my needs. **X**

Question - 3. How intuitive did you find using Search Box and Sorting Options?

Not intuitive at all.

Not very intuitive.

Somewhat intuitive.

Mostly intuitive.

Extremely intuitive. **X**

QA Test Plan

Test objectives:

The objective is to test the correctness and robustness of the search feature in the **MentorMe** product. The central idea was to regressively test the search with various filters and text searches from an end-user point of view. The testing methodologies employed was manual and postman-driven automation.

We used industry-standard testing techniques –

1. Smoke Testing
2. Regression Testing
3. Load Testing

Hardware Setup:

Linux based dual-core machine with 2GB RAM on Microsoft Azure Instance.

Software Setup:

1. Node package
2. MySQL Server
3. MySQL Client
4. Nginx Proxy Server
5. Postman (For Testing)

URL: http://20.113.70.224

Feature to be tested: The search feature was taken into consideration while applying the various testing methods.

Test Suite:

Sr. No.	Test Description	Input	Expected Output	Actual Output	Result	Google Chrome	Mozilla Firefox
1.	Responsiveness	Resolution-IpadMini	Screen Adapt to size	Screen adapted to size	Passed	✔	✔
2.	Responsiveness	Resolution – Macbook Pro	Screen Adapt to size	Screen adapted to size	Passed	✔	✔
3.	Case sensitive Tutor Search	Machine LeARning	List tutors with machine learning	Listed tutors with machine learning	Passed	✔	✔
4.	Empty search bar	Blank input	Show entire list	Showed entire list.	Passed	✔	✔
5.	Sorting - Rating	Sort Criteria – Rating	Sorted rate for tutors	Sorted rate for tutors	Passed	✔	✔
6.	Sorting - Price	Sort Criteria – Price	Sorted price for tutors	Sorted price for tutors	Passed	✔	✔
7.	Regression testing	Taking different builds on deployment to see breakage	Search shouldn't break	Search works for last 3 builds	Passed	✔	✔
8.	Load Testing	Multiple client search parallelly	Search shouldn't break	Search works	Passed	✔	✔
9.	Basic Smoke test	Searching for various subjects and tutor names	Tutor list	Tutor list fetched	Passed	✔	✔

Results:

All the various test cases drafted were found to be working as expected on the subject of test. Some of the results are formulated below for reference.

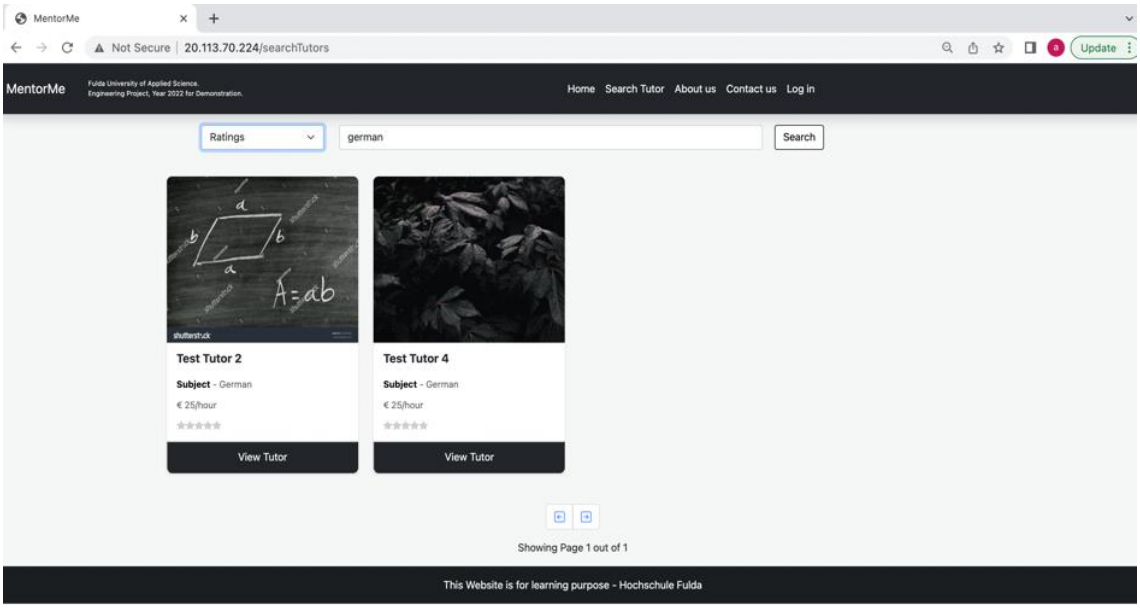

Figure-1: Search tutors teaching German, sort by Ratings

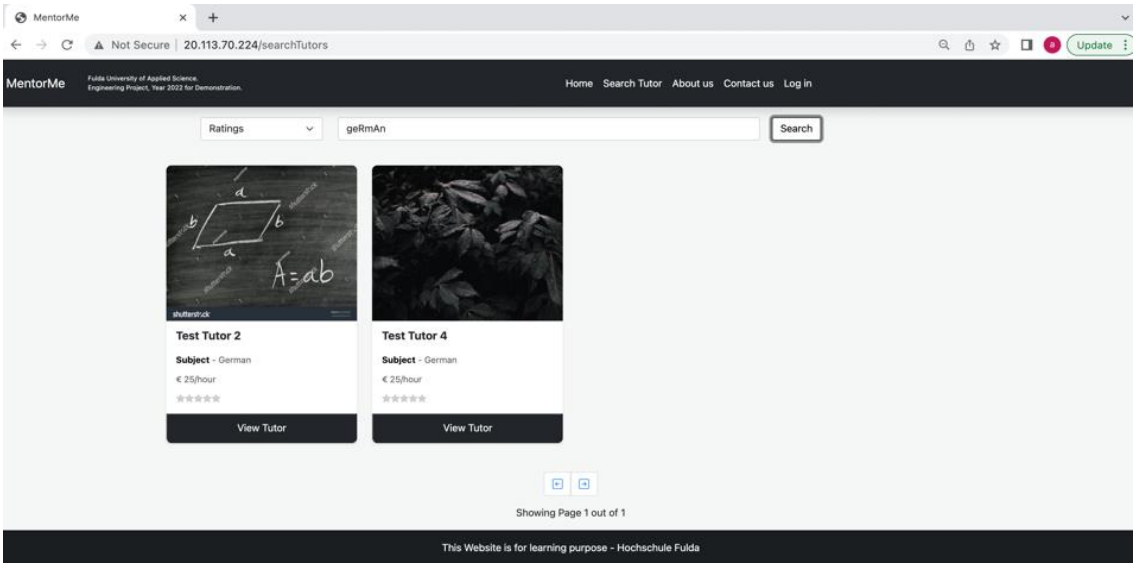

Figure-2: Search tutors teaching German case-sensitive, sort by Ratings

Code Review

Code Review by Afwan

App.js

```
//Code Reviewed by Mohammed Afwan
//Github username: theafwan
//University email: mohammed.afwan@informatik.hs-fulda.de


const express = require("express");
const cors = require("cors");
const app = express();
const dbConnection = require("./db");
const db = require("./db");
const indexRoute = require("./routes/index");
const messageController = require("./controllers/messageController");
const notificationController = require("./controllers/notificationController");
const http = require("http");


app.use(cors());
app.use(express.json());
app.use(express.static("public"));


app.use("/", indexRoute);
//*************** It's better to remove unnecessary Code*********************************
let Server = http.createServer(app);
// const io = require("socket.io")(Server, {
// cors: {
// origin: `*`,
// methods: ["GET",a "POST"],
// },
// });


//******************** On this file, It would be better to create a separate file for socket instead of on main app configuration file.**********************
//*********** You could create a function which would initiate and accept websocket connections and use all the events over there.************************
const io = require("socket.io")(Server, {
cors: {
origin: `${process.env.FRONTEND_PORT}`,
methods: ["GET", "POST"],
credentials: true,
transports: ["websocket", "polling"],
},
allowEIO3: true,
});
```

```
let users = [];


io.on("connection", (socket) => {
//login
socket.on("username", (data) => {
users.push({
id: socket.id,
userId: data.userId,
});


let len = users.length;
len--;
io.emit("userList", users, users[len].id);
});


socket.on("sendnotification", (data) => {
notificationController.sendNotificationSocket(data, (result) => {
for (let i = 0; i < users.length; i++) {
if (users[i].userId === data.tutorId) {
let socketid = users[i].id;
io.to(socketid).emit("noti", [result]);
}
}
});
});


socket.on("getnotification", (data) => {
notificationController.getNotificationSocket(data, (result) => {
for (let i = 0; i < users.length; i++) {
if (users[i].userId === data.tutorId) {
let socketid = users[i].id;
io.to(socketid).emit("noti", result);
}
}
});
});


//logout
socket.on("logout", (data) => {
for (let i = 0; i < users.length; i++) {
if (users[i].id === data.id) {
users.splice(i, 1);
}
```

```
}
io.emit("exit", users);
});
socket.on("sendmessage", async (data) => {
messageController.sendMessageSocket(data, (result) => {
io.emit("output", [result]);
});
});
socket.on("findmessage", async (data) => {
messageController.getMessageScoket(data, (result) => {
socket.emit("output", result);
});
});
});


Server.listen(4000, () => {
console.log("Server started");
});
```

Messagecontroller

```
//Code Reviewed by Mohammed Afwan
//Github username: theafwan
//University email: mohammed.afwan@informatik.hs-fulda.de


const dbConnection = require("../db");
const util = require("util");
var _ = require("underscore");
module.exports.sendMessageRequest = async (req, res) => {
let { studentId, tutorId, message } = req.body;


const addConnection = `INSERT INTO CONNECTIONS (STUDENT_ID,TUTOR_ID,REMARK,MESSAGE) VALUES
(${studentId},${tutorId},0,'${message}')`;


dbConnection.query(addConnection, async (err, result) => {
if (err) {
return res.status(400).json(err);
}
res.status(200).json({ message: "request sent" });
});
};
//********************** Describe properly what params are doing and what does the significance of value mentioned
in the comment *****************
//pending= 0, accept=1, reject=2
module.exports.changeMessageRequestStatus = async (req, res) => {
```

```
let { status, studentId, tutorId } = req.body;
const dbPromise = util.promisify(dbConnection.query).bind(dbConnection);


let sqlGetCurrentStatus = `SELECT c.REMARK FROM CONNECTIONS c WHERE c.STUDENT_ID = ${studentId} AND
c.TUTOR_ID = ${tutorId}`;
let currentStatus = null;
try {
currentStatus = await dbPromise(sqlGetCurrentStatus);
} catch (err) {
throw err;
}
if (_.isEmpty(currentStatus)) {
res.status(400).json({ message: "Connection not found" });
return;
} else {
if (currentStatus[0].REMARK === 1) {
res.status(400).json({ message: "Connection Already Accepted" });
return;
}
}


const changeStatus = `UPDATE CONNECTIONS SET REMARK=${status} WHERE STUDENT_ID=${studentId} AND
TUTOR_ID=${tutorId}`;
try {
await dbPromise(changeStatus);
} catch (err) {
throw err;
}


if (status == 1) {
const getMessageFromConn = `Select * FROM CONNECTIONS WHERE STUDENT_ID=${studentId} AND
TUTOR_ID=${tutorId}`;
let result = null;
try {
result = await dbPromise(getMessageFromConn);
} catch (err) {
throw err;
}
if (_.isEmpty(result)) {
res.status(400).json({ message: "Connection not found" });
return;
}
var connection = JSON.parse(JSON.stringify(result[0]));
console.log("Connection receieved", connection);
const pushInMessageTable = `INSERT INTO MESSAGING (SENDER_ID,RECIEVER_ID,MESSAGE) VALUES
(${studentId},${tutorId},'${connection.MESSAGE}')`;
try {
```

```
await dbPromise(pushInMessageTable);
} catch (err) {
throw err;
}
res.status(201).json({ message: "Connection Established" });
} else {
res.status(200).json({ message: "Connection Not Established" });
}
};


module.exports.getAllMessages = async (req, res) => {
let { studentId, tutorId } = req.query;
const getMessage = `SELECT * FROM MESSAGING WHERE (SENDER_ID = ${studentId} AND RECIEVER_ID = ${tutorId}) OR
(SENDER_ID = ${tutorId} AND RECIEVER_ID = ${studentId}) ORDER BY SENT_AT ASC`;
dbConnection.query(getMessage, async (err, result) => {
if (err) {
console.log(err);
}
res.status(200).json({ result });
});
};


module.exports.getAllConnections = async (req, res) => {};


module.exports.checkConnections = async (req, res) => {
let { studentId, tutorId } = req.body;


const chkConnection = `SELECT * FROM CONNECTIONS c WHERE c.STUDENT_ID = ${studentId} AND c.TUTOR_ID =
${tutorId}`;


dbConnection.query(chkConnection, async (err, result) => {
if (err) {
return res.status(200).json(err);
}
const data = JSON.parse(JSON.stringify(result));
if (_.isEmpty(data)) {
res.status(404).json({ message: "Connection not made yet." });
} else {
res.status(200).json({ remark: data[0].REMARK });
}
});
};


module.exports.sendMessageSocket = (data, callback) => {
```

```
let { receiverId, senderId, message } = data;
let createMessage = `INSERT INTO MESSAGING (SENDER_ID,RECIEVER_ID,MESSAGE) VALUES
(${senderId},${receiverId},'${message}')`;
dbConnection.query(createMessage, (err, result) => {
if (err) {
console.log(err);
}
let fetchMessage = `SELECT MESSAGE_ID,SENT_AT, UPDATED_DATE FROM MESSAGING WHERE SENDER_ID=${senderId}
AND RECIEVER_ID=${receiverId} ORDER BY SENT_AT DESC LIMIT 1`;
dbConnection.query(fetchMessage, (err, result) => {
if (err) {
console.log(err);
}
let timestamp = JSON.parse(JSON.stringify(result));
timestamp = timestamp[0];
callback({
receiverId,
senderId,
message,
messageId: timestamp.MESSAGE_ID,
timestamp: {
sentAt: timestamp.SENT_AT,
updatedAt: timestamp.UPDATED_DATE,
},
});
return;
});


// return cb(result);
});
};
//****************************************** Please add comments what this method is doing
*****************************************************
module.exports.getMessageScoket = (data, callback) => {
let { receiverId, senderId } = data;
let fetchMessage = `SELECT * FROM MESSAGING WHERE (RECIEVER_ID = ${receiverId} AND SENDER_ID=${senderId}) OR
(RECIEVER_ID=${senderId} AND SENDER_ID=${receiverId}) ORDER BY SENT_AT LIMIT 100`;
dbConnection.query(fetchMessage, (err, result) => {
if (err) {
console.log(err);
}
let message = JSON.parse(JSON.stringify(result));
callback(message);
return;
});
};
//**************** What is the roleId and it's value means?? ****************************
module.exports.getMessagingList = async (req, res) => {
let { userId, roleId } = req.query;
```

```
const dbPromise = util.promisify(dbConnection.query).bind(dbConnection);


let contacts = null;
let sqlGetContacts = null;


if (roleId == 2) {
sqlGetContacts = `SELECT STUDENT_ID FROM CONNECTIONS WHERE TUTOR_ID = ${userId} AND REMARK = 1`;
} else if (roleId == 3) {
sqlGetContacts = `SELECT TUTOR_ID FROM CONNECTIONS WHERE STUDENT_ID = ${userId} AND REMARK = 1`;
}


try {
contacts = await dbPromise(sqlGetContacts);
} catch (err) {
throw err;
}
let contactedIds = [];
for (contact of contacts) {
if (roleId == 2) {
contactedIds.push(contact.STUDENT_ID);
} else if (roleId == 3) {
contactedIds.push(contact.TUTOR_ID);
}
}


let response = [];
for (contactedId of contactedIds) {
let sqlGetContactDetails = `SELECT u.NAME,u.IMAGE FROM USER u WHERE u.USER_ID = ${contactedId} `;
let sqlGetLastMessage = `SELECT MESSAGE,SENT_AT FROM MESSAGING WHERE (SENDER_ID = ${userId} AND
RECIEVER_ID = ${contactedId}) OR (SENDER_ID = ${contactedId} AND RECIEVER_ID = ${userId}) ORDER BY SENT_AT DESC
LIMIT 1`;
let contactDetails = null;
let lastMessageDetails = null;
//********************** Instead of multiple try catch it could have been better to use single try
catch*******************
try {
contactDetails = await dbPromise(sqlGetContactDetails);
} catch (err) {
throw err;
}


try {
lastMessageDetails = await dbPromise(sqlGetLastMessage);
} catch (err) {
throw err;
```

```
}
//**************************Remove unnecessary console logs**************************
console.log(contactDetails);
console.log(lastMessageDetails);
let userDetail = {
userId: contactedId,
userName: contactDetails[0].NAME,
profilePicture: contactDetails[0].IMAGE,
timestamp: lastMessageDetails[0].SENT_AT,
lastMessage: lastMessageDetails[0].MESSAGE,
};


response.push(userDetail);
}


res.json(response);
};
```

Code Review by Mohit Dalal

File Reviewed: SearchTutor.js (Frontend File)

File Owner: Pratikkumar Kakadiya

```
    // I really like the naming convention for variables used here, it improves
the code readability.
    const [tutors, setTutors] = useState([]);
    const [searchTerm, setsearchTerm] = useState("");
    const [sortBy, setsortBy] = useState("default");
    const [page, setPage] = useState({ currentPage: 1, step: 12, numberOfPage: 0,
data: [] });
    let navigate = useNavigate();
```

```
    // The idea of making the major Axios calls of the page in a separate
function is really efficient and save lines of code,
    // it also sets all the states in a single function call, helping to update
the complete page in a single function call.
    const loadData = (sT, sB) => {
        axios.get(`${process.env.REACT_APP_SERVER_URL}/api/search/tutors?searchTe
rm=${sT}&sortBy=${sB}`)
            .then(response => {
                setTutors(response.data);
```

```
                setPage({ ...page, numberOfPage: Math.ceil(response.data.length /
page.step), data: response.data.slice(0, page.step), currentPage: 1 });

            })
            .catch(err => {
                console.log(err);
            })
    }
```

```
    const renderRating = (n) => {
        if (n === 0) {
            return (
                <div className="small-ratings">
                    <i className="bi bi-star-fill"></i>
                    <i className="bi bi-star-fill"></i>
                    <i className="bi bi-star-fill"></i>
                    <i className="bi bi-star-fill"></i>
                    <i className="bi bi-star-fill"></i>
                </div>
            )
        }
        if (n === 1) {
            return (
                <div className="small-ratings">
                    <i className="bi bi-star-fill rating-color"></i>
                    <i className="bi bi-star-fill"></i>
                    <i className="bi bi-star-fill"></i>
                    <i className="bi bi-star-fill"></i>
                    <i className="bi bi-star-fill"></i>
                </div>
            )
        }
        if (n === 2) {
            return (
                <div className="small-ratings">
                    <i className="bi bi-star-fill rating-color"></i>
                    <i className="bi bi-star-fill rating-color"></i>
                    <i className="bi bi-star-fill"></i>
                    <i className="bi bi-star-fill"></i>
                    <i className="bi bi-star-fill"></i>
                </div>
            )
        }
```

```
        if (n === 3) {
            return (
                <div className="small-ratings">
                    <i className="bi bi-star-fill rating-color"></i>
                    <i className="bi bi-star-fill rating-color"></i>
                    <i className="bi bi-star-fill rating-color"></i>
                    <i className="bi bi-star-fill"></i>
                    <i className="bi bi-star-fill"></i>
                </div>
            )
        }
        if (n === 4) {
            return (
                <div className="small-ratings">
                    <i className="bi bi-star-fill rating-color"></i>
                    <i className="bi bi-star-fill rating-color"></i>
                    <i className="bi bi-star-fill rating-color"></i>
                    <i className="bi bi-star-fill rating-color"></i>
                    <i className="bi bi-star-fill"></i>
                </div>
            )
        }
        if (n === 5) {
            return (
                <div className="small-ratings">
                    <i className="bi bi-star-fill rating-color"></i>
                    <i className="bi bi-star-fill rating-color"></i>
                    <i className="bi bi-star-fill rating-color"></i>
                    <i className="bi bi-star-fill rating-color"></i>
                    <i className="bi bi-star-fill rating-color"></i>
                </div>
            )
        }
        // The above functionality could also have been implemented in a single
block of code
        // by using ternary operators for each rating star, like following code
of block. To save
        // lines of code
        // return (<div className="small-ratings">
        //     {n >= 1 ? <i className="bi bi-star-fill rating-color"></i> : <i
className="bi bi-star-fill"></i>}
        //     {n >= 2 ? <i className="bi bi-star-fill rating-color"></i> : <i
className="bi bi-star-fill"></i>}
        //     {n >= 3 ? <i className="bi bi-star-fill rating-color"></i> : <i
className="bi bi-star-fill"></i>}
```

```
        //     {n >= 4 ? <i className="bi bi-star-fill rating-color"></i> : <i
className="bi bi-star-fill"></i>}
        //     {n >= 5 ? <i className="bi bi-star-fill rating-color"></i> : <i
className="bi bi-star-fill"></i>}
        // </div>);
    }
```

Code Review by Ankit

/*

@author **Omar Ibrahim**

@reviewer **Ankit Anand**

1. The method name is intuitive enough explaining the purpose.
2. Graceful handling of the asynchronous operation.
3. Some of the variables name could have been improved.
4. The method could also be decomposed into further routines to make it more clean and re-usable.
5. While accessing the array by index, first the size should have been checked in order to avoid any array Index out of bound exception.

*/

```
module.exports.getTutorDetails = async (req, res) => {
  let { user_id } = req.query;
  let sql = `SELECT u.NAME,u.HAS_PERMISSION,u.IMAGE,u.EMAIL,u.MOBILE_NO,u.BIO,u.REGISTERED_AT,u.GENDER,t.*FROM
TUTOR t INNER JOIN USER u ON (u.USER_ID = t.USER_ID) WHERE u.USER_ID = ${user_id}`;

  const dbPromise = util.promisify(dbConnection.query).bind(dbConnection);

  let result = null;
  try {
    result = await dbPromise(sql);
  } catch (err) {
    throw err;
  }

  if (_.isEmpty(result)) {
    res.status(400).json({ message: "Tutor Not Found" });
    return;
```

```
  }
  var tutor = JSON.parse(JSON.stringify(result[0]));
  tutor["subjects"] = [];
  let sqlSubectQuery = `SELECT s.SUBJECT_ID,s.SUBJECT_NAME, s.PRICE , AVG(r.RATING) AS AVERAGE_RATING FROM TUTOR t
INNER JOIN SUBJECT s  ON (t.USER_ID = s.USER_ID) INNER JOIN REVIEWS r ON (t.USER_ID = r.TO_USER_ID AND r.SUBJECT_ID
= s.SUBJECT_ID) WHERE t.USER_ID = ${user_id} GROUP BY s.SUBJECT_NAME`;
  let subjects = null;
  try {
    subjects = await dbPromise(sqlSubectQuery);
  } catch (err) {
    throw err;
  }
  tutor["subjects"] = JSON.parse(JSON.stringify(subjects));

  tutor["reviews"] = [];
  let sqlReviewQuery = `SELECT r.REVIEW, r.RATING FROM TUTOR t  INNER JOIN REVIEWS r  ON (t.USER_ID = r.TO_USER_ID)
WHERE t.USER_ID = ${user_id} ORDER BY r.RATING DESC`;
  let reviews = null;
  try {
    reviews = await dbPromise(sqlReviewQuery);
  } catch (err) {
    throw err;
  }
  for (review of reviews) {
    if (review.REVIEW == null) {
      review.REVIEW = "";
    }
  }
  tutor["reviews"] = JSON.parse(JSON.stringify(reviews));

  res.json(tutor);
};
```

Snippet-1

/*
@author **Omar Ibrahim**
@reviewer **Ankit Anand**

1. The method name is intuitive, explaining the purpose.
2. Graceful handling of the asynchronous database operation.
3. Graceful handling of the try-catch block.
4. The method could also be decomposed into further routines to make it more clean and re-usable.
5. Good use of constants to declare sql query.

*/

```
module.exports.getReviewOptions = async (req, res) => {
  let { studentId, tutorId } = req.body;
  const dbPromise = util.promisify(dbConnection.query).bind(dbConnection);

  let sqlIfReviewed = `SELECT ID FROM REVIEWS WHERE FROM_USER_ID =${studentId} AND
TO_USER_ID=${tutorId} `;
  let result = null;
  try {
    result = await dbPromise(sqlIfReviewed);
  } catch (err) {
    throw err;
  }
  let isReviewed = !_.isEmpty(result);
  let sqlIfContacted = `SELECT * FROM CONNECTIONS WHERE STUDENT_ID = ${studentId} AND TUTOR_ID
=${tutorId} AND REMARK = 1 `;
  try {
    result = await dbPromise(sqlIfContacted);
  } catch (err) {
    throw err;
  }
  let isContacted = !_.isEmpty(result);
  let flag = null;
  if (!isReviewed && isContacted) {
    flag = 1;
  } else {
    flag = 0;
  }
  res.send({ flag });
 };
```

Snippet-2

Code Review by Bibek

SearchController

```
const { json } = require("express");
const dbConnection = require("../db");
const util = require("util");


//Code Reviewed by Bibek Gaihre
//Github username: bibekgaihre
//University email: bibek.gaihre@informatik.hs-fulda.de
```

```
//***************On the function name, you can make the function name more clearly. For eg: a Uniform approach such as using camelCase for function name. ***************
//***************Add Comments on each major functionality.***************
//
module.exports.search_tutor_get = async (req, res) => {
let { searchTerm, sortBy } = req.query;
if (!searchTerm) {
searchTerm = "";
}
//***************Separating concern of each condition in different modules/functions. ***************
// ***************For eg: Sorting and searching concern could be decoupled into different functions if they don't have dependencies with each other. ***************


//***************Instead of callback usage, a more readable async/await from es6 can be used. There is one library to promisify the query. for example util.promisify(dbConnection.query).bind(dbConnection);***************
if (Object.keys(req.query).length === 0) {
let sql = `SELECT u.NAME,u.IMAGE,t.USER_ID , s.SUBJECT_NAME, s.PRICE , AVG(r.RATING) AS AVERAGE_RATING, s.SUBJECT_ID FROM TUTOR t INNER JOIN SUBJECT s ON (t.USER_ID = s.USER_ID) INNER JOIN USER u ON (t.USER_ID = u.USER_ID) INNER JOIN REVIEWS r ON (u.USER_ID = r.TO_USER_ID AND r.SUBJECT_ID = s.SUBJECT_ID) WHERE (t.IS_APPROVED = 1 AND u.HAS_PERMISSION=1) GROUP BY s.SUBJECT_NAME,u.NAME`;
dbConnection.query(sql, (err, result) => {
if (err) throw err;
res.send(result);
});
} // ***************Instead of using Equality operator use of "${String}.length" can be more readable.***************
else if (sortBy == "default" && searchTerm != "") {
let sql = `SELECT u.NAME,u.IMAGE,t.USER_ID , s.SUBJECT_NAME, s.PRICE , AVG(r.RATING) AS AVERAGE_RATING, s.SUBJECT_ID FROM TUTOR t INNER JOIN SUBJECT s ON (t.USER_ID = s.USER_ID) INNER JOIN USER u ON (t.USER_ID = u.USER_ID) INNER JOIN REVIEWS r ON (u.USER_ID = r.TO_USER_ID AND r.SUBJECT_ID = s.SUBJECT_ID) WHERE ((u.NAME LIKE "%${searchTerm}%" OR s.SUBJECT_NAME LIKE "%${searchTerm}%") AND t.IS_APPROVED = 1 AND u.HAS_PERMISSION=1) GROUP BY s.SUBJECT_NAME,u.NAME`;
dbConnection.query(sql, (err, result) => {
if (err) throw err;
res.send(result);
});
} else if (sortBy === "ratings" && searchTerm != "") {
let sql = `SELECT u.NAME,u.IMAGE,t.USER_ID , s.SUBJECT_NAME, s.PRICE , AVG(r.RATING) AS AVERAGE_RATING, s.SUBJECT_ID FROM TUTOR t INNER JOIN SUBJECT s ON (t.USER_ID = s.USER_ID) INNER JOIN USER u ON (t.USER_ID = u.USER_ID) INNER JOIN REVIEWS r ON (u.USER_ID = r.TO_USER_ID AND r.SUBJECT_ID = s.SUBJECT_ID) WHERE ((u.NAME LIKE "%${searchTerm}%" OR s.SUBJECT_NAME LIKE "%${searchTerm}%") AND t.IS_APPROVED = 1 AND u.HAS_PERMISSION=1) GROUP BY s.SUBJECT_NAME,u.NAME ORDER BY AVERAGE_RATING DESC`;
dbConnection.query(sql, (err, result) => {
if (err) throw err;
res.send(result);
});
} else if (sortBy === "price" && searchTerm != "") {
let sql = `SELECT u.NAME,u.IMAGE,t.USER_ID , s.SUBJECT_NAME, s.PRICE , AVG(r.RATING) AS AVERAGE_RATING, s.SUBJECT_ID FROM TUTOR t INNER JOIN SUBJECT s ON (t.USER_ID = s.USER_ID) INNER JOIN USER u ON (t.USER_ID =
```

```
u.USER_ID) INNER JOIN REVIEWS r ON (u.USER_ID = r.TO_USER_ID AND r.SUBJECT_ID = s.SUBJECT_ID) WHERE ((u.NAME
LIKE "%${searchTerm}%" OR s.SUBJECT_NAME LIKE "%${searchTerm}%") AND t.IS_APPROVED = 1 AND
u.HAS_PERMISSION=1) GROUP BY s.SUBJECT_NAME,u.NAME ORDER BY s.PRICE ASC`;
dbConnection.query(sql, (err, result) => {
if (err) throw err;
res.send(result);
});
} else if (sortBy == "default" && searchTerm == "") {
let sql = `SELECT u.NAME,u.IMAGE,t.USER_ID , s.SUBJECT_NAME, s.PRICE , AVG(r.RATING) AS AVERAGE_RATING,
s.SUBJECT_ID FROM TUTOR t INNER JOIN SUBJECT s ON (t.USER_ID = s.USER_ID) INNER JOIN USER u ON (t.USER_ID =
u.USER_ID) INNER JOIN REVIEWS r ON (u.USER_ID = r.TO_USER_ID AND r.SUBJECT_ID = s.SUBJECT_ID) WHERE
(t.IS_APPROVED = 1 AND u.HAS_PERMISSION=1) GROUP BY s.SUBJECT_NAME,u.NAME`;
dbConnection.query(sql, (err, result) => {
if (err) throw err;
res.send(result);
});
} else if (sortBy === "ratings" && searchTerm == "") {
let sql = `SELECT u.NAME,u.IMAGE,t.USER_ID , s.SUBJECT_NAME, s.PRICE , AVG(r.RATING) AS AVERAGE_RATING,
s.SUBJECT_ID FROM TUTOR t INNER JOIN SUBJECT s ON (t.USER_ID = s.USER_ID) INNER JOIN USER u ON (t.USER_ID =
u.USER_ID) INNER JOIN REVIEWS r ON (u.USER_ID = r.TO_USER_ID AND r.SUBJECT_ID = s.SUBJECT_ID) WHERE
(t.IS_APPROVED = 1 AND u.HAS_PERMISSION=1) GROUP BY s.SUBJECT_NAME,u.NAME ORDER BY AVERAGE_RATING
DESC`;
dbConnection.query(sql, (err, result) => {
if (err) throw err;
res.send(result);
});
} else if (sortBy === "price" && searchTerm == "") {
let sql = `SELECT u.NAME,u.IMAGE,t.USER_ID , s.SUBJECT_NAME, s.PRICE , AVG(r.RATING) AS AVERAGE_RATING,
s.SUBJECT_ID FROM TUTOR t INNER JOIN SUBJECT s ON (t.USER_ID = s.USER_ID) INNER JOIN USER u ON (t.USER_ID =
u.USER_ID) INNER JOIN REVIEWS r ON (u.USER_ID = r.TO_USER_ID AND r.SUBJECT_ID = s.SUBJECT_ID) WHERE
(t.IS_APPROVED = 1 AND u.HAS_PERMISSION=1) GROUP BY s.SUBJECT_NAME,u.NAME ORDER BY s.PRICE ASC;`;
dbConnection.query(sql, (err, result) => {
if (err) throw err;
res.send(result);
});
} else {
let sql = `SELECT u.NAME,u.IMAGE,t.USER_ID , s.SUBJECT_NAME, s.PRICE , AVG(r.RATING) AS AVERAGE_RATING,
s.SUBJECT_ID FROM TUTOR t INNER JOIN SUBJECT s ON (t.USER_ID = s.USER_ID) INNER JOIN USER u ON (t.USER_ID =
u.USER_ID) INNER JOIN REVIEWS r ON (u.USER_ID = r.TO_USER_ID AND r.SUBJECT_ID = s.SUBJECT_ID) WHERE (u.NAME
LIKE "%${searchTerm}%" OR s.SUBJECT_NAME LIKE "%${searchTerm}%" WHERE t.IS_APPROVED = 1 AND
u.HAS_PERMISSION=1) GROUP BY s.SUBJECT_NAME,u.NAME`;
dbConnection.query(sql, (err, result) => {
if (err) throw err;
res.send(result);
});
}
};
```

Code Review by Ahmed

```
import { useState, useEffect } from "react";
import axios from "axios";
import { useCookies } from "react-cookie";
import { useNavigate } from "react-router-dom";
import $ from 'jquery';
import 'datatables.net';

const PendingRequest = () => {
    const [tutors, setTutors] = useState([]);
    const [cookies, setCookie] = useCookies(['user']);
    const navigate = useNavigate();

    const loadData = () => {
        axios.get(`${process.env.REACT_APP_SERVER_URL}/api/notVerifiedTutors`, { headers: {
"Authorization": `Bearer ${cookies.token}` } })
            .then((response) => {
                setTutors(response.data);
                $(document).ready(function () {
                    $('#pendingTutors').dataTable({
                        responsive: true,
                    });
                });
            })
            .catch((error) => {
                console.log(console.error);
            });
    }

    useEffect(() => {
        loadData();
    }, []);

    const handleClick = (e) => {
        e.preventDefault();
        navigate(`/viewTutorProfile/${e.currentTarget.id}`)
    }
    const renderDate = (d) => {
        const registrationDate = new Date(d);
        return (
            registrationDate.getDate().toString() + "." +
```

```
            (registrationDate.getMonth() + 1).toString() + "." +
            registrationDate.getFullYear().toString()
        );
    }

    return (
        <>
            <link rel="stylesheet" type="text/css"
href="https://cdn.datatables.net/1.11.4/css/jquery.dataTables.css"></link>
            <div className="container mt-3">
                <div className="row">
                    <div className="container mt-1 mb-3">
                        <h3>New Tutors</h3>
                    </div>
                </div>
                <hr />
                <div className="card">
                    <div className="card-body">
                        {tutors.length > 0 ?
                            <table className="table table-hover mt-1" id="pendingTutors">
                                <thead>
                                    <tr>
                                        <th scope="col"></th>
                                        <th scope="col">Name</th>
                                        <th scope="col">Requested Date</th>
                                        <th></th>
                                    </tr>
                                </thead>
                                <tbody>
                                    {
                                        tutors.map(tutor => (

                                            <tr key={tutor.USER_ID}>
                                                <td>
                                                    <img
src={`${process.env.REACT_APP_PROFILE_URL}${tutor.IMAGE}`} style={{ width: "65px",
height: "65px", borderRadius: "50%" }} alt="avatar" />
                                                </td>
                                                <td>
                                                    {tutor.NAME}
                                                </td>
                                                <td>
                                                     {renderDate(tutor.UPDATED_DATE)}
                                                </td>
```

```
                                    <td>
                                        <button className="btn btn-outline-dark"
onClick={handleClick} id={tutor.USER_ID}>View Profile</button>
                                    </td>
                                </tr>

                            ))
                        }
                    </tbody>
                </table> :
                <div className="card-body">
                    <div className="container d-flex justify-content-center my-5">
                        <h3>No Pending Requests</h3>
                    </div>
                </div>
            }
        </div>
    </div>

    </div>
  </>
  )

}
export default PendingRequest;
```

//****************** Ahmad Estaitia Review ****************/
// The code is understandable and organized, but it would have been better to put the comments to be clearer and easier to read
// The code has some deprecated function.

Code Reviewed By - Pratikkumar A. Kakadiya

Author - Mohit Dalal

File - TutorSignUp.js

```
                    >
                      {confirmPassEmpty}
                    </span>
                  </div>
                </div>
                <div className="mt-5 text-center">
                  <button
                    className="btn btn-primary profile-button"
                    type="submit"
                    onClick={submit}
                  >
                    Sign Up
                  </button>
                </div>
              </div>
            </div>
          </div>
        </form>
      </div>
    </>
  );
};
export default TutorSignUp;

// -------------------- Reviewed by Pratikkumar A. Kakadiya --------------------
// -> In my opinion onstead of using separate state for each error it could be combined in one state as key-value pair
// it will increase code readability and maintainability both.
// -> Code for validating form could be separated as function to further improve maintainability.
```

Code Review by Omar

1- rejectProfileWithReason

```
/*
* Code Review Basic Authentication is not needed as token validation and role checking
* are done in the middleware
* The name of the functional is intuitive and does not need further explanation,
* also camel case is used which is the standard
*/
module.exports.rejectProfileWithReason = (req, res) => {
  /* Code Review
  *  As only a moderator can reject a profile, it would make more sense to name senderId as moderatorId
  *  and receiverId as rejectedUserId. same changes should apply to the table column names also.
  */
```

```
  let { reason, senderId, receiverId } = req.body;
  const postRejectMessage = `INSERT INTO REJECT_REASON ( REASON,
SENDER_ID,RECEIVER_ID,TIME_SENT ) VALUES
("${reason}",${senderId},${receiverId},"${new Date()
    .toISOString()
    .replace(/T/, " ")
    .replace(/\..+/, "")}")`;

  /* Code Review
  *  using async for the query callback function seems like an overkill and
unnecessary
  *  for the error handling, the err variable should be checked to identify the
error
  *  and send a message of what the error is and also depending on the error the
HTTP
  *  error code may change
  */
  dbConnection.query(postRejectMessage, async (err, result) => {
    if (err) {
      return res.status(400).json(err);
    }
    res.status(200).json(result);
  });
};
```

2- banProfile

```
/**
 * Code Review function name is intuitive, it is async as Promises are used for
db queries
 * because multiple queries are called to avoid callback hell which is good
 */
module.exports.banProfile = async (req, res) => {
  /**
   * Code Review variable names are intuitive and need not further explanation
   * BANNED_USER table would be better if SENDER_ID was replaced with
MODERATOR_ID
   * and RECEIVER_ID with USER_ID
   */
  let { reason, moderatorId, userId } = req.body;
  const dbPromise = util.promisify(dbConnection.query).bind(dbConnection);
  const updateUser = `UPDATE User
                      SET HAS_PERMISSION = 0
                      WHERE USER_ID = ${userId}`;
  const postRejectMessage = `INSERT INTO
                             BANNED_USER ( REASON,
SENDER_ID,RECEIVER_ID,TIME_SENT )
                             VALUES ("${reason}",${moderatorId},${userId},"${new
Date()
      .toISOString()
      .replace(/T/, " ")
      .replace(/\..+/, "")}")`;

  /**
   * Code Review
   * error handling could be handlet better by creating a custom Error object
with the error message
   */
  try {
    result = await dbPromise(updateUser);
  } catch (err) {
    throw err;
  }

  try {
    result = await dbPromise(postRejectMessage);
  } catch (err) {
    throw err;
  }
  res.status(200).json(result);
```

```
};
```

Code Review by Yogeeta

Code Reviewed By – Yogeeta Sharma
Author – Ahmed
File - TutorSignUp.js

```
                                </div>
                            </div>
                        </div>
                    </div>
                ))
                :
                <div className="card-body">
                    <div className="container d-flex justify-content-center my-5">
                        <h3>No Pending Messaging Requests</h3>
                    </div>
                </div>
            }
        </div>
    )
}

export default ViewMessageRequest;

// ***************************************   Review by Yogeeta  ***************************************************
// It should have been a better practice to use comments when using fucntions for better understanding of the code.
```

Self-check on best practices for security

Major Assets to Protect

1. **User Content**
    - Profile Information
    - Passwords
    - Profile Image
2. **Tutor Specific Content**
    - CV
    - Student Reviews and Ratings (Tutor Profile Info.)
    - Subjects (Tutor Profile Info.)
    - Bio (Tutor Profile Info.)
3. **Chats**

Major Threats on Assets and Protection Mechanism

1. **User Content**
    - Profile Information: Only the user with valid credentials should be able to access this information. To protect this we have implemented login.
    - Passwords: If the database gets exposed or breached, then illegal parties can access the credentials. In order to protect this we are doing one way hashing and then storing it in the Database.
    - Profile Image: Profile image could be collected by some illegal/unauthorized party, to protect this we are just saving randomly generated unique name of the image in the database, and uploading real data to the secure Azure cloud.
2. **Tutor Specific Content**
    - CV: CV File could be collected by some illegal/unauthorized party, to protect this we are just saving randomly generated unique names of the file in the database, and uploading real data to the secure Azure cloud.
3. **Chats**
    - Chats Should be one to one and no other user should be able to access others chats, we are maintaining one to one chatting by uniquely identifying each message with the help of userID.

We are encrypting Passwords in the Database!

- We are validating the input data on each and every form, for empty fields and field specific information with the help of regular expression string comparison.

Self-check: Adherence to original Non-functional specs – performed by team leads

- Application shall be developed, tested and deployed using tools and servers approved by Class CTO and as agreed in Milestone 0. Application delivery shall be from chosen cloud server. **Done**
- Application shall be optimized for standard desktop/laptop browsers e.g. must render correctly on the two latest versions of two major browsers **Done**
- All or selected application functions must render well on mobile devices **Done**
- Data shall be stored in the database on the team's deployment cloud server. **Done**
- No more than 50 concurrent users shall be accessing the application at any time **Done**
- Privacy of users shall be protected and all privacy policies will be appropriately communicated to the users. **Done**
- The language used shall be English (no localization needed) **Done**
- Application shall be very easy to use and intuitive **Done**
- Application should follow established architecture patterns **Done**
- Application code and its repository shall be easy to inspect and maintain **Done**
- Google analytics shall be used (optional for Fulda teams) **NA**
- No e-mail clients shall be allowed. **Done**
- Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI. **Done**
- Site security: basic best practices shall be applied (as covered in the class) for main data items **Done**
- Application shall be media rich (images, video etc.). Media formats shall be standard as used in the market today **Done**
- Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development **Done**
- For code development and management, as well as documentation like formal milestones required in the class, each team shall use their own GitHub to be set-up by class instructors and started by each team during Milestone 0 **Done**
- The application UI (WWW and mobile) shall prominently display the following exact text on all pages "Fulda University of Applied Sciences Software Engineering

Project, Fall 2021 For Demonstration Only” at the top of the WWW page. (Important so as to not confuse this with a real application). **Done**