

Master Team Project WS2021

Milestone 4: HelpMeLearn

Master Team Project Winter 2021
Global Distributed Software Development

CEO & CTO:

Prof. Dr. Rainer Todtenhöfer

Team 04

Hasib Iqbal (hasib.iqbal@informatik.hs-fulda.de)	:	Team Lead Frontend Developer
Mohammad Rakibul Hasan	:	Frontend Lead
Rohat Sagar Urif Sonu	:	Frontend Developer
Mohammad Salman Haydar	:	Backend Lead
Talha Jahangiri Khan	:	Github Lead Backend Developer
Chowdhury Amlan Barua	:	Cloud & Backend Developer
Nisha Devi	:	Backend Developer

Revision	Date
Version 1.0	11 March 2022

11 March 2022

Table of Contents

1	Product Summary
2	Usability Test Plan
3	QA Test Plan
4	Code Review
5	Self Check on best practices for security
6	Self-check: Adherence to original Non-functional specs

Product Summary

Motivation & Importance

Humanity has become accustomed to clicks as the most expedient method of obtaining what one desires. In this age of clicks, we want to make life easy for students and teachers alike. However, we have chosen to focus on one of everyone's most basic requirements, which is learning. HelpMeLearn is a web-based tutoring service that connects students and teachers.

Functions & Services

Our product aims to establish a communication bridge between tutor and students. From the perspective of students, they can search for tutors for their desired subjects using our platform and also can communicate with them. Our platform provides real time chat facilities which can help the students to get connected with the tutors instantly. They can also share their experience about tutors through our review system. From the perspective of the tutor, they can upload their CV, qualification and show their skills through our tutor profile page to get the interests of the students. They can mention their hourly salary expectations, reply to messages from the students using our platform. Also, every change made by the tutor in their timeline is reviewed and approved by the admin before going live to maintain the standard of our platform. Our platform also provides a guest page where a guest user can search for their desired subjects and tutors without registering. The only catch here is that they cannot communicate with the tutor. Our searching page also provides facilities about filtering different categories like by subject, gender, etc.

Reasons to use our HelpMeLearn Application

Using our HelpMeLearn application for learning will make students and teachers connect in the most efficient way without having to pay a great sum of money for subscription and we are ensuring a safe platform for both users. If you are a student or teacher, it creates a safe platform for a teacher and a student to connect as the opportunity to offer courses by a teacher is endless and students can easily compare the offered courses.

Major Committed Functions:

1. Search courses

- Allowing/Enabling the students to find certain approved courses that satisfy / fulfill a criterion.

2. Filter searched courses

- Specific attributes a student can use to refine the search results or course. e.g., by gender, price range etc.

3. Add /View Reviews

- Allowing/Enabling the students to view reviews of teachers that can help students find suitable teachers

4. Registration

- Allowing the students and teachers to make a new account profile for themselves.

5. Login

- Allowing the students and teachers to log in to their account or dashboard.

6. Post Courses

- Allowing the approved teachers to add new courses from their profiles.

7. Messaging

- Allowing the students and teachers to communicate with each other through messages and also enables them to see the previous message history as well from their dashboard.

8. Approve/Reject courses / teacher information

- Allowing the admins to approve / reject different pending courses or personal information of teachers from their dashboard.

Usability Test Plan

This document describes a test plan for conducting a usability test during the development of HelpMeLearn. The goals of usability testing include establishing a baseline of user performance, establishing, and validating user performance measures, and identifying potential design concerns to be addressed to improve the efficiency, productivity, and end user satisfaction.

Test objectives

The objective of this testing is to get the user feedback and user experience about searching for a specific tutor using searched keywords, subject filter or choosing specific subject level.

Test background and setup

- **System setup**

HelpMeLearn website have been published on AWS to be tested by participants using the following technologies

- Operating System: Ubuntu 18.04 Server
- Database: MySQL v: 8.0
- Web Server: NGINX 1.18
- Server-Side Language: Node.js

- **Starting point**

Participants will test HelpMeLearn website using their laptops or smartphones, and will focus on the search functionality in the home page

- **The intended users**

The test will be conducted with a targeted category of users, which are the students and tutors of Fulda university. Therefore, we will ask for users of the university to test the website

- **URL of the system to be tested**

HelpMeLearn (the link may change due to redeploying)

- **What is to be measured**

We will be focusing on user satisfaction

Usability Testing Phase:

Test Participant 1:

Phase 1: Screening and Pretest:

Q1. How old are you?

24

Q2. What is the highest level of education you've completed?

Bachelor's Degree

Q3. What is your current occupation?

Student(Pursuing Masters)

Q4. On a scale of 1 to 5 how would you rate your level of confidence in using your laptop/pc for looking for a tutor online?

5

Q5. When was the last time you looked for a tutoring service online?

In 2020

Phase: 2 Usability Task description

Task 1: Imagine that you are facing difficulty in studying statistics and want to get help from an expert.

Task 2: Imagine that your exams are near and facing difficulty in solving exercises of economics

Task 3: Imagine you are looking for a tutor in fulda and worried about feedback from other students.

Phase: 3 Post Test Questions

Q1. How was your overall experience when searching for a tutor?

Good

Q2. How simple and clean was the interface?

It is very simple.

Q3. Can you tell me what you think about filtering the tutor by subject name and level?

I tried multiple times and it worked without any issues.

Q4. How was your experience looking for tutor qualifications and feedback?

Simple and clear.

Sl. No	Task Objective	Scale 0/5 (0-OK, 5-Good)
Task 1	Search for a tutor online by subject name	4/5
Task 2	Search for a tutor online by subject level	4/5
Task 3	Looking for tutor feedback	3/5

Test Participant 2:

Phase 1: Screening and Pretest:

Q1. How old are you?

19

Q2. What is the highest level of education you've completed?

College

Q3. What is your current occupation?

Student(Pursuing Bachelors)

Q4. On a scale of 1 to 5 how would you rate your level of confidence in using your laptop/pc for looking for a tutor online?

5

Q5. When was the last time you looked for a tutoring service online?

In 2020

Phase: 2 Usability Task description

Task 1: Imagine that you are facing difficulty in studying statistics and want to get help from an expert.

Task 2: Imagine that your exams are near and facing difficulty in solving exercises of economics

Task 3: Imagine you are looking for a tutor in fulda and worried about feedback from other students.

Phase: 3 Post Test Questions

Q1. How was your overall experience when searching for a tutor?

Good and simple.

Q2. How simple and clean was the interface?

It is very simple and easy to use.

Q3. Can you tell me what you think about filtering the tutor by subject name and level?
Filters are clear and easy to use.

Q4. How was your experience looking for tutor qualifications and feedback?
Clearly mentioned.

Sl. No	Task Objective	Scale 0/5 (0-OK, 5-Good)
Task 1	Search for a tutor online by subject name	4/5
Task 2	Search for a tutor online by subject level	3/5
Task 3	Looking for tutor feedback	4/5

Test Participant 3:

Phase 1: Screening and Pretest:

Q1. How old are you?
21

Q2. What is the highest level of education you've completed?
University

Q3. What is your current occupation?
Student(Pursuing Bachelors in Natural Science)

Q4. On a scale of 1 to 5 how would you rate your level of confidence in using your laptop/pc for looking for a tutor online?
4

Q5. When was the last time you looked for a tutoring service online?
In 2019

Phase: 2 Usability Task description

Task 1: Imagine that you are facing difficulty in studying statistics and want to get help from an expert.

Task 2: Imagine that your exams are near and facing difficulty in solving exercises of economics

Task 3: Imagine you are looking for a tutor in fulda and worried about feedback from other students.

Phase: 3 Post Test Questions

Q1. How was your overall experience when searching for a tutor?

Great and amazing.

Q2. How simple and clean was the interface?

Interface is very easy to use. And I did not have any issue adjusting to the interface.

Q3. Can you tell me what you think about filtering the tutor by subject name and level?

Filters are clear and easy to use. However, I would appreciate more filtering options.

Q4. How was your experience looking for tutor qualifications and feedback?

Clearly mentioned.

Sl. No	Task Objective	Scale 0/5 (0-OK, 5-Good)
Task 1	Search for a tutor online by subject name	4/5
Task 2	Search for a tutor online by subject level	4/5
Task 3	Looking for tutor feedback	3/5

QA Test Plan

Test Objective:

The test objectives are to verify the functionality of website HelpMeLearn and to ensure that the software is as per business requirements by identifying any bugs or issues and fixing them before release.

Hardware and Software setup:

Laptop or smartphone using Edge, Google Chrome or Firefox browser.

Feature to be tested:

- Search By Subject Name
- Search By Subject Level
- Search By Tutor Gender

Test Case Id & Browser	Test Title	Test Description	Test Input	Expected Result	Test Result
TC_1 (Chrome, Firefox)	View all tutors	Students are able to view all tutors in a list view.	Login with valid credentials. Students will be navigated to the Search Page	The Search Page displays all approved tutors.	PASS
TC_2 (Chrome, Firefox)	Filter Tutors by subject name	Students are able to filter tutors by subject name.	Input ' Math101 ' on SubjectName text field and click on the Search button	Students are able to see all approved tutors who teach the Math101 subject.	PASS
TC_3 (Chrome, Firefox)	Filter Tutors by subject level.	Students are able to filter tutors by subject level.	Select ' Master ' in the subject level drop down field and click on the Search button.	Students are able to see all approved tutors who teach at least one subject with a master's level.	PASS
TC_4 (Chrome, Firefox)	Filter Tutors by gender.	Students are able to filter tutors by gender.	Select ' Male ' in the Gender drop down field and click on the Search button.	Students are able to see all approved tutors whose gender is male.	PASS

TC_5 (Chrome, Firefox)	Filter Tutors by subject Name, level, and gender.	Students are able to filter tutors by subject name, level and gender.	Input ' Math101 ' in the Subject Name Field, select ' Master ' in the subject level drop down field, select ' Male ' in the gender drop down field and click on the Search button.	Students are able to see all approved tutors who teach SubjectName 'Math101' with subject level 'Master' and gender is male.	PASS
------------------------------	------------------------------------------------------------	--------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------	------

Code Review

Developer	Rohat Sagar Urif Sonu
Reviewer	Nisha Devi
<p>File: socketIO.js</p> <pre> 33 io.on("connection", (socket) => { 34 /* Review: Remove console.log and add try and catch and log the exception if occur. */ 35 console.log("client connected"); 36 37 socket.on("connectUser", async (payload) => { 38 const { userId } = payload; 39 40 if (userId === undefined (await fetchUser(userId)) === undefined) 41 return; 42 43 addConnectedUser(userId, socket.id); 44 45 // emit previous texts 46 socket.emit("userTextsFetched", await fetchUserTexts(userId)); 47 }); 48 49 socket.on("sendText", async (payload) => { 50 const { from, to, text } = payload; 51 52 if (from === undefined to === undefined) return; 53 54 const fromUser = await fetchUser(from); 55 56 const toUser = await fetchUser(to); 57 58 if (fromUser === undefined toUser === undefined) return; </pre>	

```
JS socketIO.js M X
C: > git > tutoringServiceGDSD > server > socketIO > JS socketIO.js > <unknown> > exports > io.on("connection") callback > socket.on("connectUser") callback
103 | /* Review: Rename to fetchUserById */
104 | async function fetchUser(userID) {
105 |     const query =
106 |         "SELECT user.*, CONCAT(user.firstName, ' ', user.lastName) as userName FROM hm_user user WHERE user.id = ?";
107 |     const queryParams = [userID];
108 |     var result = await executeQuery(query, queryParams);
109 |     return result.length !== 0 ? result[0] : undefined;
110 | }
111 |
112 | async function insertUserText(fromUserId, toUserId, text, date) {
113 |     const query =
114 |         "INSERT INTO hm_chat (fromUserId, toUserId, text, createdAt, msgStatus) VALUES (?, ?, ?, ?, ?)";
115 |     const queryParams = [fromUserId, toUserId, text, date, 1];
116 |
117 |     /* Review: Remove affectedRows since it is not used any where.*/
118 |     var { affectedRows, insertId } = await executeQuery(query, queryParams);
119 |     return insertId;
120 | }
121 |
122 | /* Review: Rename to fetchUsersById */
123 | async function fetchUserTexts(userID) {
124 |     const query = `SELECT
125 |         chat.*,
126 |         CONCAT(toUser.firstName, ' ', toUser.lastName) as toUserName,
127 |         CONCAT(fromUser.firstName, ' ', fromUser.lastName) as fromUserName
128 |     FROM hm_chat chat`
```

File: Chat.js

```
src > components > chat > JS Chat.js > Chat
60
61 | export default function Chat(props: Props) {
62 |     /*
63 |     | Review: Remove the hardcoded image url and place in the common file.
64 |     */
65 |     const imageUrl = "logo512.png";
66 |
67 |     const { showChat, chatClosed, selectedUserId } = props;
68 |
69 |     const socket = useRef();
70 |     const textControl = useRef();
71 |     const [texts, setTexts] = useState([]);
72 |     const [arrivalMessage, setArrivalMessage] = useState(null);
73 |     const [selectedChat, setSelectedChat] = useState(
74 |         getDefaultSelectedChat(selectedUserId, texts)
75 |     );
76 |     const currentUser = useSelector(getCurrentUser);
77 |
78 |     useEffect(() => {
79 |         if (arrivalMessage) {
80 |             const { userID, userName, id, date, text, inbox } = arrivalMessage;
81 |
82 |             let recipientIndex = texts.findIndex((text) => text.userID == userID);
83 |
84 |             if (recipientIndex === -1) {
85 |                 let newItem = {
```

Developer	Nisha Devi
Reviewer	Rohat Sagar Urif Sonu

File: postController.js

```

JS postController.js > ...
1  let database = require("../database");
2  const { validationResult } = require("express-validator");
3  const util = require("util");
4
5  const executeQuery = util.promisify(database.query).bind(database);
6
7  module.exports = {
8    createPost: async (req, res) => {
9      const errors = validationResult(req);
10     if (!errors.isEmpty()) {
11       return res.status(400).json({ errors: errors.array() });
12     }
13
14     // ** COMMENT: View can follow the same naming convention as rest in the code e.g. camelCase
15     let {
16       Description,
17       Status,
18       Language,
19       SubjectName,
20       RatePerHour,
21       ExperinceYears,
22       AvailableTime,
23       UserId,
24     } = req.body;
25
26     // ** COMMENT: Can we move this to a util function?
27     var date = new Date().toISOString().split("T")[0];
28     var isActive = true;
29
30     try {
31       let result = await executeQuery(
32         "SELECT * FROM hm_tutor_profile T WHERE T.userId = ?;",
33         [userId]
34       );
35
36       if (result.length > 0) {
37         return res.status(400).json({ errors: errors.array() });
38       }
39
40       let result = await executeQuery(
41         "INSERT INTO hm_post (description, tutorProfileId, status, language, subjectName, ratePerHour, modifiedDateTime, experinceYears, availableTime, id) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?);",
42         [
43           Description,
44           tutorProfileId,
45           Status,
46           Language,
47           SubjectName,
48           RatePerHour,
49           date,
50           ExperinceYears,
51           AvailableTime,
52           Id,
53         ]
54       );
55
56       if (result.length > 0) {
57         return res.status(201).json({ message: "Post created successfully" });
58       }
59
60       return res.status(400).json({ errors: errors.array() });
61     } catch (error) {
62       console.error(error);
63       return res.status(500).json({ message: "Internal server error" });
64     }
65   },
66
67   updatePost: async (req, res) => {
68     const errors = validationResult(req);
69     if (!errors.isEmpty()) {
70       return res.status(400).json({ errors: errors.array() });
71     }
72
73     let {
74       Id,
75       Description,
76       TutorProfileId,
77       Status,
78       Language,
79       SubjectName,
80       RatePerHour,
81       ExperinceYears,
82       AvailableTime,
83     } = req.body;
84
85     // ** COMMENT: Can we move this to a util function?
86     var date = new Date().toISOString().split("T")[0];
87
88     database.query(
89       "UPDATE hm_post SET description=?, tutorProfileId=?, status=?, `language`=?, subjectName=?, ratePerHour=?, modifiedDateTime=?, experinceYe
90       [
91         Description,
92         TutorProfileId,
93         Status,
94         Language,
95         SubjectName,
96         RatePerHour,
97         date,
98         ExperinceYears,
99         AvailableTime,
100        Id,
101      ],
102      (err, result) => {
103        if (err) {
104          console.error(err);
105          return res.status(500).json({ message: "Internal server error" });
106        }
107
108        if (result.length > 0) {
109          return res.status(200).json({ message: "Post updated successfully" });
110        }
111
112        return res.status(400).json({ errors: errors.array() });
113      }
114    );
115  },
116
117   deletePost: async (req, res) => {
118     const errors = validationResult(req);
119     if (!errors.isEmpty()) {
120       return res.status(400).json({ errors: errors.array() });
121     }
122
123     let { Id } = req.body;
124
125     database.query(
126       "DELETE FROM hm_post WHERE id = ?",
127       [Id]
128     );
129
130     return res.status(200).json({ message: "Post deleted successfully" });
131   },
132
133   getAllPosts: async (req, res) => {
134     database.query(
135       "SELECT * FROM hm_post",
136       (err, result) => {
137         if (err) {
138           console.error(err);
139           return res.status(500).json({ message: "Internal server error" });
140         }
141
142         return res.status(200).json(result);
143       }
144     );
145   },
146
147   getPostById: async (req, res) => {
148     const errors = validationResult(req);
149     if (!errors.isEmpty()) {
150       return res.status(400).json({ errors: errors.array() });
151     }
152
153     let { Id } = req.params;
154
155     database.query(
156       "SELECT * FROM hm_post WHERE id = ?",
157       [Id]
158     );
159
160     return res.status(200).json(result);
161   },
162
163   getPostByTutorProfileId: async (req, res) => {
164     const errors = validationResult(req);
165     if (!errors.isEmpty()) {
166       return res.status(400).json({ errors: errors.array() });
167     }
168
169     let { tutorProfileId } = req.params;
170
171     database.query(
172       "SELECT * FROM hm_post WHERE tutorProfileId = ?",
173       [tutorProfileId]
174     );
175
176     return res.status(200).json(result);
177   },
178
179   getPostBySubjectName: async (req, res) => {
180     const errors = validationResult(req);
181     if (!errors.isEmpty()) {
182       return res.status(400).json({ errors: errors.array() });
183     }
184
185     let { subjectName } = req.params;
186
187     database.query(
188       "SELECT * FROM hm_post WHERE subjectName = ?",
189       [subjectName]
190     );
191
192     return res.status(200).json(result);
193   },
194
195   getPostByLanguage: async (req, res) => {
196     const errors = validationResult(req);
197     if (!errors.isEmpty()) {
198       return res.status(400).json({ errors: errors.array() });
199     }
200
201     let { language } = req.params;
202
203     database.query(
204       "SELECT * FROM hm_post WHERE language = ?",
205       [language]
206     );
207
208     return res.status(200).json(result);
209   },
210
211   getPostByRatePerHour: async (req, res) => {
212     const errors = validationResult(req);
213     if (!errors.isEmpty()) {
214       return res.status(400).json({ errors: errors.array() });
215     }
216
217     let { ratePerHour } = req.params;
218
219     database.query(
220       "SELECT * FROM hm_post WHERE ratePerHour = ?",
221       [ratePerHour]
222     );
223
224     return res.status(200).json(result);
225   },
226
227   getPostByExperinceYears: async (req, res) => {
228     const errors = validationResult(req);
229     if (!errors.isEmpty()) {
230       return res.status(400).json({ errors: errors.array() });
231     }
232
233     let { experinceYears } = req.params;
234
235     database.query(
236       "SELECT * FROM hm_post WHERE experinceYears = ?",
237       [experinceYears]
238     );
239
240     return res.status(200).json(result);
241   },
242
243   getPostByAvailableTime: async (req, res) => {
244     const errors = validationResult(req);
245     if (!errors.isEmpty()) {
246       return res.status(400).json({ errors: errors.array() });
247     }
248
249     let { availableTime } = req.params;
250
251     database.query(
252       "SELECT * FROM hm_post WHERE availableTime = ?",
253       [availableTime]
254     );
255
256     return res.status(200).json(result);
257   },
258
259   getPostByStatus: async (req, res) => {
260     const errors = validationResult(req);
261     if (!errors.isEmpty()) {
262       return res.status(400).json({ errors: errors.array() });
263     }
264
265     let { status } = req.params;
266
267     database.query(
268       "SELECT * FROM hm_post WHERE status = ?",
269       [status]
270     );
271
272     return res.status(200).json(result);
273   },
274
275   getPostByUserId: async (req, res) => {
276     const errors = validationResult(req);
277     if (!errors.isEmpty()) {
278       return res.status(400).json({ errors: errors.array() });
279     }
280
281     let { userId } = req.params;
282
283     database.query(
284       "SELECT * FROM hm_post WHERE userId = ?",
285       [userId]
286     );
287
288     return res.status(200).json(result);
289   },
290
291   getPostByIsActive: async (req, res) => {
292     const errors = validationResult(req);
293     if (!errors.isEmpty()) {
294       return res.status(400).json({ errors: errors.array() });
295     }
296
297     let { isActive } = req.params;
298
299     database.query(
300       "SELECT * FROM hm_post WHERE isActive = ?",
301       [isActive]
302     );
303
304     return res.status(200).json(result);
305   },
306
307   getPostByModifiedDate: async (req, res) => {
308     const errors = validationResult(req);
309     if (!errors.isEmpty()) {
310       return res.status(400).json({ errors: errors.array() });
311     }
312
313     let { modifiedDateTime } = req.params;
314
315     database.query(
316       "SELECT * FROM hm_post WHERE modifiedDateTime = ?",
317       [modifiedDateTime]
318     );
319
320     return res.status(200).json(result);
321   },
322
323   getPostByDescription: async (req, res) => {
324     const errors = validationResult(req);
325     if (!errors.isEmpty()) {
326       return res.status(400).json({ errors: errors.array() });
327     }
328
329     let { description } = req.params;
330
331     database.query(
332       "SELECT * FROM hm_post WHERE description = ?",
333       [description]
334     );
335
336     return res.status(200).json(result);
337   },
338
339   getPostBySubjectNameAndLanguage: async (req, res) => {
340     const errors = validationResult(req);
341     if (!errors.isEmpty()) {
342       return res.status(400).json({ errors: errors.array() });
343     }
344
345     let { subjectName, language } = req.params;
346
347     database.query(
348       "SELECT * FROM hm_post WHERE subjectName = ? AND language = ?",
349       [subjectName, language]
350     );
351
352     return res.status(200).json(result);
353   },
354
355   getPostBySubjectNameAndRatePerHour: async (req, res) => {
356     const errors = validationResult(req);
357     if (!errors.isEmpty()) {
358       return res.status(400).json({ errors: errors.array() });
359     }
360
361     let { subjectName, ratePerHour } = req.params;
362
363     database.query(
364       "SELECT * FROM hm_post WHERE subjectName = ? AND ratePerHour = ?",
365       [subjectName, ratePerHour]
366     );
367
368     return res.status(200).json(result);
369   },
370
371   getPostBySubjectNameAndExperinceYears: async (req, res) => {
372     const errors = validationResult(req);
373     if (!errors.isEmpty()) {
374       return res.status(400).json({ errors: errors.array() });
375     }
376
377     let { subjectName, experinceYears } = req.params;
378
379     database.query(
380       "SELECT * FROM hm_post WHERE subjectName = ? AND experinceYears = ?",
381       [subjectName, experinceYears]
382     );
383
384     return res.status(200).json(result);
385   },
386
387   getPostBySubjectNameAndAvailableTime: async (req, res) => {
388     const errors = validationResult(req);
389     if (!errors.isEmpty()) {
390       return res.status(400).json({ errors: errors.array() });
391     }
392
393     let { subjectName, availableTime } = req.params;
394
395     database.query(
396       "SELECT * FROM hm_post WHERE subjectName = ? AND availableTime = ?",
397       [subjectName, availableTime]
398     );
399
400     return res.status(200).json(result);
401   },
402
403   getPostBySubjectNameAndStatus: async (req, res) => {
404     const errors = validationResult(req);
405     if (!errors.isEmpty()) {
406       return res.status(400).json({ errors: errors.array() });
407     }
408
409     let { subjectName, status } = req.params;
410
411     database.query(
412       "SELECT * FROM hm_post WHERE subjectName = ? AND status = ?",
413       [subjectName, status]
414     );
415
416     return res.status(200).json(result);
417   },
418
419   getPostBySubjectNameAndIsActive: async (req, res) => {
420     const errors = validationResult(req);
421     if (!errors.isEmpty()) {
422       return res.status(400).json({ errors: errors.array() });
423     }
424
425     let { subjectName, isActive } = req.params;
426
427     database.query(
428       "SELECT * FROM hm_post WHERE subjectName = ? AND isActive = ?",
429       [subjectName, isActive]
430     );
431
432     return res.status(200).json(result);
433   },
434
435   getPostBySubjectNameAndModifiedDate: async (req, res) => {
436     const errors = validationResult(req);
437     if (!errors.isEmpty()) {
438       return res.status(400).json({ errors: errors.array() });
439     }
440
441     let { subjectName, modifiedDateTime } = req.params;
442
443     database.query(
444       "SELECT * FROM hm_post WHERE subjectName = ? AND modifiedDateTime = ?",
445       [subjectName, modifiedDateTime]
446     );
447
448     return res.status(200).json(result);
449   },
450
451   getPostBySubjectNameAndDescription: async (req, res) => {
452     const errors = validationResult(req);
453     if (!errors.isEmpty()) {
454       return res.status(400).json({ errors: errors.array() });
455     }
456
457     let { subjectName, description } = req.params;
458
459     database.query(
460       "SELECT * FROM hm_post WHERE subjectName = ? AND description = ?",
461       [subjectName, description]
462     );
463
464     return res.status(200).json(result);
465   },
466
467   getPostBySubjectNameAndSubjectNameAndLanguage: async (req, res) => {
468     const errors = validationResult(req);
469     if (!errors.isEmpty()) {
470       return res.status(400).json({ errors: errors.array() });
471     }
472
473     let { subjectName, language } = req.params;
474
475     database.query(
476       "SELECT * FROM hm_post WHERE subjectName = ? AND language = ?",
477       [subjectName, language]
478     );
479
480     return res.status(200).json(result);
481   },
482
483   getPostBySubjectNameAndSubjectNameAndRatePerHour: async (req, res) => {
484     const errors = validationResult(req);
485     if (!errors.isEmpty()) {
486       return res.status(400).json({ errors: errors.array() });
487     }
488
489     let { subjectName, ratePerHour } = req.params;
490
491     database.query(
492       "SELECT * FROM hm_post WHERE subjectName = ? AND ratePerHour = ?",
493       [subjectName, ratePerHour]
494     );
495
496     return res.status(200).json(result);
497   },
498
499   getPostBySubjectNameAndSubjectNameAndExperinceYears: async (req, res) => {
500     const errors = validationResult(req);
501     if (!errors.isEmpty()) {
502       return res.status(400).json({ errors: errors.array() });
503     }
504
505     let { subjectName, experinceYears } = req.params;
506
507     database.query(
508       "SELECT * FROM hm_post WHERE subjectName = ? AND experinceYears = ?",
509       [subjectName, experinceYears]
510     );
511
512     return res.status(200).json(result);
513   },
514
515   getPostBySubjectNameAndSubjectNameAndAvailableTime: async (req, res) => {
516     const errors = validationResult(req);
517     if (!errors.isEmpty()) {
518       return res.status(400).json({ errors: errors.array() });
519     }
520
521     let { subjectName, availableTime } = req.params;
522
523     database.query(
524       "SELECT * FROM hm_post WHERE subjectName = ? AND availableTime = ?",
525       [subjectName, availableTime]
526     );
527
528     return res.status(200).json(result);
529   },
530
531   getPostBySubjectNameAndSubjectNameAndStatus: async (req, res) => {
532     const errors = validationResult(req);
533     if (!errors.isEmpty()) {
534       return res.status(400).json({ errors: errors.array() });
535     }
536
537     let { subjectName, status } = req.params;
538
539     database.query(
540       "SELECT * FROM hm_post WHERE subjectName = ? AND status = ?",
541       [subjectName, status]
542     );
543
544     return res.status(200).json(result);
545   },
546
547   getPostBySubjectNameAndSubjectNameAndIsActive: async (req, res) => {
548     const errors = validationResult(req);
549     if (!errors.isEmpty()) {
550       return res.status(400).json({ errors: errors.array() });
551     }
552
553     let { subjectName, isActive } = req.params;
554
555     database.query(
556       "SELECT * FROM hm_post WHERE subjectName = ? AND isActive = ?",
557       [subjectName, isActive]
558     );
559
560     return res.status(200).json(result);
561   },
562
563   getPostBySubjectNameAndSubjectNameAndModifiedDate: async (req, res) => {
564     const errors = validationResult(req);
565     if (!errors.isEmpty()) {
566       return res.status(400).json({ errors: errors.array() });
567     }
568
569     let { subjectName, modifiedDateTime } = req.params;
570
571     database.query(
572       "SELECT * FROM hm_post WHERE subjectName = ? AND modifiedDateTime = ?",
573       [subjectName, modifiedDateTime]
574     );
575
576     return res.status(200).json(result);
577   },
578
579   getPostBySubjectNameAndSubjectNameAndDescription: async (req, res) => {
580     const errors = validationResult(req);
581     if (!errors.isEmpty()) {
582       return res.status(400).json({ errors: errors.array() });
583     }
584
585     let { subjectName, description } = req.params;
586
587     database.query(
588       "SELECT * FROM hm_post WHERE subjectName = ? AND description = ?",
589       [subjectName, description]
590     );
591
592     return res.status(200).json(result);
593   },
594
595   getPostBySubjectNameAndSubjectNameAndSubjectNameAndLanguage: async (req, res) => {
596     const errors = validationResult(req);
597     if (!errors.isEmpty()) {
598       return res.status(400).json({ errors: errors.array() });
599     }
600
601     let { subjectName, language } = req.params;
602
603     database.query(
604       "SELECT * FROM hm_post WHERE subjectName = ? AND language = ?",
605       [subjectName, language]
606     );
607
608     return res.status(200).json(result);
609   },
610
611   getPostBySubjectNameAndSubjectNameAndSubjectNameAndRatePerHour: async (req, res) => {
612     const errors = validationResult(req);
613     if (!errors.isEmpty()) {
614       return res.status(400).json({ errors: errors.array() });
615     }
616
617     let { subjectName, ratePerHour } = req.params;
618
619     database.query(
620       "SELECT * FROM hm_post WHERE subjectName = ? AND ratePerHour = ?",
621       [subjectName, ratePerHour]
622     );
623
624     return res.status(200).json(result);
625   },
626
627   getPostBySubjectNameAndSubjectNameAndSubjectNameAndExperinceYears: async (req, res) => {
628     const errors = validationResult(req);
629     if (!errors.isEmpty()) {
630       return res.status(400).json({ errors: errors.array() });
631     }
632
633     let { subjectName, experinceYears } = req.params;
634
635     database.query(
636       "SELECT * FROM hm_post WHERE subjectName = ? AND experinceYears = ?",
637       [subjectName, experinceYears]
638     );
639
640     return res.status(200).json(result);
641   },
642
643   getPostBySubjectNameAndSubjectNameAndSubjectNameAndAvailableTime: async (req, res) => {
644     const errors = validationResult(req);
645     if (!errors.isEmpty()) {
646       return res.status(400).json({ errors: errors.array() });
647     }
648
649     let { subjectName, availableTime } = req.params;
650
651     database.query(
652       "SELECT * FROM hm_post WHERE subjectName = ? AND availableTime = ?",
653       [subjectName, availableTime]
654     );
655
656     return res.status(200).json(result);
657   },
658
659   getPostBySubjectNameAndSubjectNameAndSubjectNameAndStatus: async (req, res) => {
660     const errors = validationResult(req);
661     if (!errors.isEmpty()) {
662       return res.status(400).json({ errors: errors.array() });
663     }
664
665     let { subjectName, status } = req.params;
666
667     database.query(
668       "SELECT * FROM hm_post WHERE subjectName = ? AND status = ?",
669       [subjectName, status]
670     );
671
672     return res.status(200).json(result);
673   },
674
675   getPostBySubjectNameAndSubjectNameAndSubjectNameAndIsActive: async (req, res) => {
676     const errors = validationResult(req);
677     if (!errors.isEmpty()) {
678       return res.status(400).json({ errors: errors.array() });
679     }
680
681     let { subjectName, isActive } = req.params;
682
683     database.query(
684       "SELECT * FROM hm_post WHERE subjectName = ? AND isActive = ?",
685       [subjectName, isActive]
686     );
687
688     return res.status(200).json(result);
689   },
690
691   getPostBySubjectNameAndSubjectNameAndSubjectNameAndModifiedDate: async (req, res) => {
692     const errors = validationResult(req);
693     if (!errors.isEmpty()) {
694       return res.status(400).json({ errors: errors.array() });
695     }
696
697     let { subjectName, modifiedDateTime } = req.params;
698
699     database.query(
700       "SELECT * FROM hm_post WHERE subjectName = ? AND modifiedDateTime = ?",
701       [subjectName, modifiedDateTime]
702     );
703
704     return res.status(200).json(result);
705   },
706
707   getPostBySubjectNameAndSubjectNameAndSubjectNameAndDescription: async (req, res) => {
708     const errors = validationResult(req);
709     if (!errors.isEmpty()) {
710       return res.status(400).json({ errors: errors.array() });
711     }
712
713     let { subjectName, description } = req.params;
714
715     database.query(
716       "SELECT * FROM hm_post WHERE subjectName = ? AND description = ?",
717       [subjectName, description]
718     );
719
720     return res.status(200).json(result);
721   },
722
723   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndLanguage: async (req, res) => {
724     const errors = validationResult(req);
725     if (!errors.isEmpty()) {
726       return res.status(400).json({ errors: errors.array() });
727     }
728
729     let { subjectName, language } = req.params;
730
731     database.query(
732       "SELECT * FROM hm_post WHERE subjectName = ? AND language = ?",
733       [subjectName, language]
734     );
735
736     return res.status(200).json(result);
737   },
738
739   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndRatePerHour: async (req, res) => {
740     const errors = validationResult(req);
741     if (!errors.isEmpty()) {
742       return res.status(400).json({ errors: errors.array() });
743     }
744
745     let { subjectName, ratePerHour } = req.params;
746
747     database.query(
748       "SELECT * FROM hm_post WHERE subjectName = ? AND ratePerHour = ?",
749       [subjectName, ratePerHour]
750     );
751
752     return res.status(200).json(result);
753   },
754
755   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndExperinceYears: async (req, res) => {
756     const errors = validationResult(req);
757     if (!errors.isEmpty()) {
758       return res.status(400).json({ errors: errors.array() });
759     }
760
761     let { subjectName, experinceYears } = req.params;
762
763     database.query(
764       "SELECT * FROM hm_post WHERE subjectName = ? AND experinceYears = ?",
765       [subjectName, experinceYears]
766     );
767
768     return res.status(200).json(result);
769   },
770
771   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndAvailableTime: async (req, res) => {
772     const errors = validationResult(req);
773     if (!errors.isEmpty()) {
774       return res.status(400).json({ errors: errors.array() });
775     }
776
777     let { subjectName, availableTime } = req.params;
778
779     database.query(
780       "SELECT * FROM hm_post WHERE subjectName = ? AND availableTime = ?",
781       [subjectName, availableTime]
782     );
783
784     return res.status(200).json(result);
785   },
786
787   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndStatus: async (req, res) => {
788     const errors = validationResult(req);
789     if (!errors.isEmpty()) {
790       return res.status(400).json({ errors: errors.array() });
791     }
792
793     let { subjectName, status } = req.params;
794
795     database.query(
796       "SELECT * FROM hm_post WHERE subjectName = ? AND status = ?",
797       [subjectName, status]
798     );
799
800     return res.status(200).json(result);
801   },
802
803   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndIsActive: async (req, res) => {
804     const errors = validationResult(req);
805     if (!errors.isEmpty()) {
806       return res.status(400).json({ errors: errors.array() });
807     }
808
809     let { subjectName, isActive } = req.params;
810
811     database.query(
812       "SELECT * FROM hm_post WHERE subjectName = ? AND isActive = ?",
813       [subjectName, isActive]
814     );
815
816     return res.status(200).json(result);
817   },
818
819   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndModifiedDate: async (req, res) => {
820     const errors = validationResult(req);
821     if (!errors.isEmpty()) {
822       return res.status(400).json({ errors: errors.array() });
823     }
824
825     let { subjectName, modifiedDateTime } = req.params;
826
827     database.query(
828       "SELECT * FROM hm_post WHERE subjectName = ? AND modifiedDateTime = ?",
829       [subjectName, modifiedDateTime]
830     );
831
832     return res.status(200).json(result);
833   },
834
835   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndDescription: async (req, res) => {
836     const errors = validationResult(req);
837     if (!errors.isEmpty()) {
838       return res.status(400).json({ errors: errors.array() });
839     }
840
841     let { subjectName, description } = req.params;
842
843     database.query(
844       "SELECT * FROM hm_post WHERE subjectName = ? AND description = ?",
845       [subjectName, description]
846     );
847
848     return res.status(200).json(result);
849   },
850
851   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndLanguage: async (req, res) => {
852     const errors = validationResult(req);
853     if (!errors.isEmpty()) {
854       return res.status(400).json({ errors: errors.array() });
855     }
856
857     let { subjectName, language } = req.params;
858
859     database.query(
860       "SELECT * FROM hm_post WHERE subjectName = ? AND language = ?",
861       [subjectName, language]
862     );
863
864     return res.status(200).json(result);
865   },
866
867   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndRatePerHour: async (req, res) => {
868     const errors = validationResult(req);
869     if (!errors.isEmpty()) {
870       return res.status(400).json({ errors: errors.array() });
871     }
872
873     let { subjectName, ratePerHour } = req.params;
874
875     database.query(
876       "SELECT * FROM hm_post WHERE subjectName = ? AND ratePerHour = ?",
877       [subjectName, ratePerHour]
878     );
879
880     return res.status(200).json(result);
881   },
882
883   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndExperinceYears: async (req, res) => {
884     const errors = validationResult(req);
885     if (!errors.isEmpty()) {
886       return res.status(400).json({ errors: errors.array() });
887     }
888
889     let { subjectName, experinceYears } = req.params;
890
891     database.query(
892       "SELECT * FROM hm_post WHERE subjectName = ? AND experinceYears = ?",
893       [subjectName, experinceYears]
894     );
895
896     return res.status(200).json(result);
897   },
898
899   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndAvailableTime: async (req, res) => {
900     const errors = validationResult(req);
901     if (!errors.isEmpty()) {
902       return res.status(400).json({ errors: errors.array() });
903     }
904
905     let { subjectName, availableTime } = req.params;
906
907     database.query(
908       "SELECT * FROM hm_post WHERE subjectName = ? AND availableTime = ?",
909       [subjectName, availableTime]
910     );
911
912     return res.status(200).json(result);
913   },
914
915   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndStatus: async (req, res) => {
916     const errors = validationResult(req);
917     if (!errors.isEmpty()) {
918       return res.status(400).json({ errors: errors.array() });
919     }
920
921     let { subjectName, status } = req.params;
922
923     database.query(
924       "SELECT * FROM hm_post WHERE subjectName = ? AND status = ?",
925       [subjectName, status]
926     );
927
928     return res.status(200).json(result);
929   },
930
931   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndIsActive: async (req, res) => {
932     const errors = validationResult(req);
933     if (!errors.isEmpty()) {
934       return res.status(400).json({ errors: errors.array() });
935     }
936
937     let { subjectName, isActive } = req.params;
938
939     database.query(
940       "SELECT * FROM hm_post WHERE subjectName = ? AND isActive = ?",
941       [subjectName, isActive]
942     );
943
944     return res.status(200).json(result);
945   },
946
947   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndModifiedDate: async (req, res) => {
948     const errors = validationResult(req);
949     if (!errors.isEmpty()) {
950       return res.status(400).json({ errors: errors.array() });
951     }
952
953     let { subjectName, modifiedDateTime } = req.params;
954
955     database.query(
956       "SELECT * FROM hm_post WHERE subjectName = ? AND modifiedDateTime = ?",
957       [subjectName, modifiedDateTime]
958     );
959
960     return res.status(200).json(result);
961   },
962
963   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndDescription: async (req, res) => {
964     const errors = validationResult(req);
965     if (!errors.isEmpty()) {
966       return res.status(400).json({ errors: errors.array() });
967     }
968
969     let { subjectName, description } = req.params;
970
971     database.query(
972       "SELECT * FROM hm_post WHERE subjectName = ? AND description = ?",
973       [subjectName, description]
974     );
975
976     return res.status(200).json(result);
977   },
978
979   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndLanguage: async (req, res) => {
980     const errors = validationResult(req);
981     if (!errors.isEmpty()) {
982       return res.status(400).json({ errors: errors.array() });
983     }
984
985     let { subjectName, language } = req.params;
986
987     database.query(
988       "SELECT * FROM hm_post WHERE subjectName = ? AND language = ?",
989       [subjectName, language]
990     );
991
992     return res.status(200).json(result);
993   },
994
995   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndRatePerHour: async (req, res) => {
996     const errors = validationResult(req);
997     if (!errors.isEmpty()) {
998       return res.status(400).json({ errors: errors.array() });
999     }
1000
1001     let { subjectName, ratePerHour } = req.params;
1002
1003     database.query(
1004       "SELECT * FROM hm_post WHERE subjectName = ? AND ratePerHour = ?",
1005       [subjectName, ratePerHour]
1006     );
1007
1008     return res.status(200).json(result);
1009   },
1010
1011   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndExperinceYears: async (req, res) => {
1012     const errors = validationResult(req);
1013     if (!errors.isEmpty()) {
1014       return res.status(400).json({ errors: errors.array() });
1015     }
1016
1017     let { subjectName, experinceYears } = req.params;
1018
1019     database.query(
1020       "SELECT * FROM hm_post WHERE subjectName = ? AND experinceYears = ?",
1021       [subjectName, experinceYears]
1022     );
1023
1024     return res.status(200).json(result);
1025   },
1026
1027   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndAvailableTime: async (req, res) => {
1028     const errors = validationResult(req);
1029     if (!errors.isEmpty()) {
1030       return res.status(400).json({ errors: errors.array() });
1031     }
1032
1033     let { subjectName, availableTime } = req.params;
1034
1035     database.query(
1036       "SELECT * FROM hm_post WHERE subjectName = ? AND availableTime = ?",
1037       [subjectName, availableTime]
1038     );
1039
1040     return res.status(200).json(result);
1041   },
1042
1043   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndStatus: async (req, res) => {
1044     const errors = validationResult(req);
1045     if (!errors.isEmpty()) {
1046       return res.status(400).json({ errors: errors.array() });
1047     }
1048
1049     let { subjectName, status } = req.params;
1050
1051     database.query(
1052       "SELECT * FROM hm_post WHERE subjectName = ? AND status = ?",
1053       [subjectName, status]
1054     );
1055
1056     return res.status(200).json(result);
1057   },
1058
1059   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndIsActive: async (req, res) => {
1060     const errors = validationResult(req);
1061     if (!errors.isEmpty()) {
1062       return res.status(400).json({ errors: errors.array() });
1063     }
1064
1065     let { subjectName, isActive } = req.params;
1066
1067     database.query(
1068       "SELECT * FROM hm_post WHERE subjectName = ? AND isActive = ?",
1069       [subjectName, isActive]
1070     );
1071
1072     return res.status(200).json(result);
1073   },
1074
1075   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndModifiedDate: async (req, res) => {
1076     const errors = validationResult(req);
1077     if (!errors.isEmpty()) {
1078       return res.status(400).json({ errors: errors.array() });
1079     }
1080
1081     let { subjectName, modifiedDateTime } = req.params;
1082
1083     database.query(
1084       "SELECT * FROM hm_post WHERE subjectName = ? AND modifiedDateTime = ?",
1085       [subjectName, modifiedDateTime]
1086     );
1087
1088     return res.status(200).json(result);
1089   },
1090
1091   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndDescription: async (req, res) => {
1092     const errors = validationResult(req);
1093     if (!errors.isEmpty()) {
1094       return res.status(400).json({ errors: errors.array() });
1095     }
1096
1097     let { subjectName, description } = req.params;
1098
1099     database.query(
1100       "SELECT * FROM hm_post WHERE subjectName = ? AND description = ?",
1101       [subjectName, description]
1102     );
1103
1104     return res.status(200).json(result);
1105   },
1106
1107   getPostBySubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndSubjectNameAndLanguage: async (req, res) => {
1108     const errors = validationResult(req);
1109     if (!errors.isEmpty()) {
1110       return res.status(400).json({ errors: errors.array() });
1111     }
1
```

```

JS postController.js > [0] <unknown> > updatePost
110 |   },
111 |   (err, result) => {
112 |     if (err) res.status(400).send(`Response Error: ${err}`);
113 |     else res.status(204).json({ message: "Post Details Updated" });
114 |   }
115 | };
116 | },
117 |
118 | getPost: async (req, res) => {
119 |   let id = req.params.id;
120 |   database.query(
121 |     "SELECT id, description, tutorProfileId, status, `language`, subjectName, ratePerHour, createdDateTime, modifiedDateTime, experienceYears,
122 |     [id],
123 |     (err, result) => {
124 |       if (err) res.status(400).send(`Response Error: ${err}`);
125 |       else res.status(200).json(result);
126 |     }
127 |   );
128 | },
129 |
130 | searchPost: async (req, res) => {
131 |
132 |   // ** COMMENT: We can move join query logic to a method.
133 |   let joinQuery = "";
134 |   if (req.query.TutorProfileId !== undefined) {
135 |     joinQuery += `tutorProfileId = ${database.escape(
136 |       req.query.TutorProfileId
137 |     )}`;
138 |   }
139 |
140 |   if (req.query.Status !== undefined) {
141 |     if (joinQuery !== "") joinQuery += " and ";
142 |

```

```

JS postController.js > [0] <unknown> > searchPost
143 |   joinQuery += `status = ${database.escape(req.query.Status)}`;
144 | }
145 |
146 | if (req.query.RatePerHour !== undefined) {
147 |   if (joinQuery !== "") joinQuery += " and ";
148 |
149 |   // ** COMMENT: We should add check for greater than or equal
150 |   joinQuery += `ratePerHour = ${database.escape(req.query.RatePerHour)}`;
151 | }
152 |
153 | if (req.query.SubjectName !== undefined) {
154 |   if (joinQuery !== "") joinQuery += " and ";
155 |
156 |   // ** COMMENT: Is this a case-sensitive?
157 |   joinQuery += `MATCH(subjectName) AGAINST (${database.escape(
158 |     req.query.SubjectName
159 |   )})`;
160 | }
161 |
162 | let dbQuery =
163 |   "SELECT hm_post.id, hm_post.description, hm_post.tutorProfileId, hm_post.status, hm_post.language, hm_post.subjectName, hm_post.ratePerHour
164 |   " INNER JOIN hm_tutor_profile ON (hm_tutor_profile.id = hm_post.tutorProfileId)" +
165 |   " INNER JOIN hm_user ON (hm_user.id = hm_tutor_profile.userId)";
166 | if (joinQuery !== "") dbQuery += ` where ${joinQuery}`;
167 |
168 | database.query(dbQuery, (err, result) => {
169 |   // ** COMMENT: In case of error I think we should return some error.
170 |   if (err) console.log(err);
171 |   else res.json(result);
172 | });
173 | },
174 | };

```

Developer	Hasib Iqbal
Reviewer	Chowdhury Amlan Barua
File Name	ManageTutorsProfile.js tutor.js

File Name: ManageTutorsProfile.js

```
import React from "react";
import { useSelector } from "react-redux";
import { ListGroup } from "react-bootstrap";
import TutorProfileItem from "../TutorProfileItem";
import { getTutorsProfileList } from
"../../../../../core/selectors/manageTutorsProfile";
import Page from "../../../../../components/page/Page";
import FilterBar from "../filterBar/FilterBar";

// Destructuring the props might be a good idea. You can do this with the
reference below:
// https://medium.com/@lcriswell/destructuring-props-in-react-b1c295005ce0

function ManageTutorsProfile(props) {
  var data = useSelector(getTutorsProfileList);

  if (data === undefined) {
    return <div></div>;
  }

  return (
    <Page>
      <FilterBar />
      <br />
      <ListGroup>
        {data?.map((item, i) => {
          return <TutorProfileItem key={i} item={item} />;
        })}
      </ListGroup>
      <br />
    </Page>
  )
}
```

```
);  
}  
  
export default ManageTutorsProfile;
```

File Name: tutor.js

```
export function* getTutorList(action: Object): Saga<void> {  
  const { filters } = action.payload;  
  
  var url = allTutorListApi;  
  
  if (filters.fName) {  
    url += `&FirstName=${filters.fName}`;  
  }  
  if (filters.lName) {  
    url += `&LastName=${filters.lName}`;  
  }  
  if (filters.email) {  
    url += `&Email=${filters.email}`;  
  }  
  
  const apiOptions: ApiOptions = {  
    url: url,  
    method: "GET",  
    useJwtSecret: false,  
  };  
  
  const apiResponse: ApiResponse = yield call(executeApiCall, apiOptions);  
  
  const { isSuccessful, response = {} } = apiResponse;  
  
  if (isSuccessful) {  
    var data = response;  
    yield put(getTutorListSuccess({ data }));  
  } else {  
    var msg = "Failed to load data from API"; //A more descriptive error  
message might be constructed
```



```

    yield put(getTutorListFailed({ msg }));
  }
}

```

Developer	Mohammad Rakibul Hasan
Reviewer	Hasib Iqbal
File Name	AddQualification.js

```

function AddQualification(props) {
  const dispatch = useDispatch();

  const subjectRef = useRef(null);
  const qualificationRef = useRef(null);
  const gradeRef = useRef(null);
  const descriptionRef = useRef(null);

  const user = useSelector(getCurrentUser);
  console.log("userid" + user );

  // Review Comment:
  // 1.Follow Javascript naming convention for variable names (Ref>
https://www.w3schools.com/js/js\_conventions.asp)
  // 2.Add comments for better code readability
  // 3. Remove unnecessary codes

  //function to save the qualification
  const submitQualification = () => {
    const qualification = {
      SubjectName: subjectRef.current.value,
      Grade: gradeRef.current.value,
      Description: descriptionRef.current.value,
      UserId: user.id
    };
  };

```

```

    console.log(qualification);
    dispatch(saveQualification(qualification));
    //test
    // dispatch(fetchQualificationById(1));
  };

  return (
    <div>
      <Page></Page>
      <div className="qualification-page">
        <div className="qualification-content">
          <h1>Add Qualification</h1>
          <Form>
            <br />
            <Form.Control type="text" ref={subjectRef} placeholder="Subject"
          />
            <br />
            { /* <Form.Control type="text" ref={qualificationRef}
placeholder="Qualification" />
            <br /> */ }
            <Form.Control type="text" ref={gradeRef} placeholder="Grade" />
            <br />
            <Form.Control
              ref={descriptionRef}
              as="textarea"
              rows={3}
              placeholder="Description"
            />
            <Button className="btn btn-success" variant="primary"
onClick={submitQualification} type="submit">
              Save
            </Button>
          </Form>
        </div>
      </div>
    </div>
  );
}

```

Developer	Mohammad Salman Haydar
Reviewer	Talha Jahangir Khan
File Name	UploadController.js

```

controller > JS uploadController.js > upload
1  const uploadFile = require("../middleware/upload");
2  const database = require("../database");
3  const util = require("util");
4  require("dotenv").config();
5
6  const executeQuery = util.promisify(database.query).bind(database);
7
8  const upload = async (req, res) => {
9    try {
10      await uploadFile(req, res);
11
12      if (req.file == undefined) {
13        return res.status(400).send({ message: "Please upload a file!" });
14      }
15
16      var result = await executeQuery('SELECT id FROM hm_tutor_profile WHERE userId = ?', [req.userid]);
17      var tutorProfileId = result[0].id;
18
19      if(req.file.mimetype === "application/pdf") {
20
21        database.execute("SELECT * FROM `helpmelearn`.`hm_file` WHERE `tutorProfileId` = ?",
22          [tutorProfileId],
23          (err, result) => {
24            if(err) {
25              console.log(err);
26              res.status(500).send({message:"Something went wrong"});
27            }
28

```

```

28      }
29      else if(result.length >= 1) {
30        database.execute("DELETE FROM `helpmelearn`.`hm_file` WHERE (`tutorProfileId` = ?)",
31          [tutorProfileId],(err, result)=> {
32            if(err) {
33              console.log(err);
34              res.status(500).send({message:"Something went wrong"});
35            }
36            else {
37              database.execute("INSERT INTO `helpmelearn`.`hm_file` ( `tutorProfileId`, `fileName`,
38                [tutorProfileId,
39                req.file.originalname,
40                0,
41                "pdf",
42                "resources/static/"+req.file.originalname],
43                (err, result) => {
44
45                  if (err){
46                    console.log(err);
47                    res.status(500).send({message: "Somethid went wrong during inserting into DB
48                  }
49
50                  res.status(200).send({
51                    message: "Uploaded the file successfully: " + req.file.originalname,
52                  });
53                });
54            }
55          });
56      }

```

```

    }
    else {

        database.execute("INSERT INTO `helpmelearn`.`hm_file` ( `tutorProfileId`, `fileName`,
        [tutorProfileId,
        req.file.originalname,
        0,
        "pdf",
        "resources/static/"+req.file.originalname],
        (err, result) => {

            if (err){
                console.log(err);
                res.status(500).send({message: "Somethid went wrong during inserting into DB"});
            }

            res.status(200).send({
                message: "Uploaded the file successfully: " + req.file.originalname,
            });
        });
    }
});
}

```

```

80 }
81 else if(req.file.mimetype === "image/jpg" || req.file.mimetype === "image/jpeg" || req.file.mimetype === "image/png") {
82
83     var today = new Date();
84     var date = today.getFullYear()+'-'+(today.getMonth()+1)+'-'+today.getDate();
85     var time = today.getHours() + ":" + today.getMinutes() + ":" + today.getSeconds();
86     var dateTime = date+' '+time;
87     console.log(req.userid);
88     database.execute("SELECT * FROM `helpmelearn`.`hm_image` WHERE `userId` = ?",
89     [req.userid],
90     (err, result) => {
91         if(err) {
92             console.log(err);
93             res.status(500).send({message:"Something went wrong"});
94         }
95         else if(result.length >= 1) {
96             database.execute("DELETE FROM `helpmelearn`.`hm_image` WHERE (`userId` = ?)",
97             [req.userid],(err, result)=> {
98                 if(err) {
99                     console.log(err);
100                     res.status(500).send({message:"Something went wrong"});
101                 }
102                 else {
103                     database.execute("INSERT INTO `helpmelearn`.`hm_image` ( `imagePath`, `date`, `userId`, `createdDateTime`,
104                     ["resources/static/"+req.file.originalname,
105                     dateTime,
106                     req.userid,
107                     dateTime,
108                     dateTime],

```

```

109         (err, result) => {
110             if (err){ console.log(err);
111                 res.status(500).send({message: "Somethid went wrong during inserting into DB"}});
112             }
113             res.status(200).send({
114                 message: "Uploaded the image successfully: " + req.file.originalname,
115                 });
116         });
117     }
118 });
119 }
120 else {
121
122     database.execute("INSERT INTO `helpmelearn`.`hm_image` ( `imagePath`, `date`, `userId`, `createdDateTime`, `modified
123     ["resources/static/"+req.file.originalname,
124     dateTime,
125     req.userid,
126     dateTime,
127     dateTime],
128     (err, result) => {
129         if (err){ console.log(err);
130             res.status(500).send({message: "Somethid went wrong during inserting into DB"}});
131         }
132         res.status(200).send({
133             message: "Uploaded the image successfully: " + req.file.originalname,
134             });
135         });
136     }
137 });
138 }

```

```

119     }
120     else {
121
122         database.execute("INSERT INTO `helpmelearn`.`hm_image` ( `imagePath`, `date`, `userId`, `createdDateTime`, `modified
123         ["resources/static/"+req.file.originalname,
124         dateTime,
125         req.userid,
126         dateTime,
127         dateTime],
128         (err, result) => {
129             if (err){ console.log(err);
130                 res.status(500).send({message: "Somethid went wrong during inserting into DB"}});
131             }
132             res.status(200).send({
133                 message: "Uploaded the image successfully: " + req.file.originalname,
134                 });
135             });
136         }
137     });
138 }
139
140 catch (err) {
141     res.status(500).send({
142         message: `Could not upload the file: ${req.file?.originalname}. ${err}`,
143     });
144 }
145
146 file.exports = {
147     "upload" : upload,
148 }

```

Developer	Chowdhury Amlan Barua
Reviewer	Hasib Iqbal
File Name	TutorprofilController.js

```

server > controller > JS TutorProfileController.js
4  require("dotenv").config();
5  const util = require("util");
6
7  const executeQuery = util.promisify(database.query).bind(database);
8
9  module.exports = {
10
11      // Review: In case of error handling, I think, we should use try catch functionality
12      getTutorAbouInfoById: async (req, res) => {
13          let id = req.params.id;
14          let query = `SELECT firstName, lastName, about, age, picPath FROM hm_tutor_profile A, hm_user B WHERE A.userId = B.id AND userI
15
16          database.query(query, [id], (err, result) => {
17              if (err) console.log(err);
18              else res.json(result);
19          });
20      },
21
22      // Review: 1. In case of error handling, I think, we should use try catch functional"ity
23      //          2. Remove console.log as it is not doing anything.
24      getTutorOfferedCoursesById: async (req, res) => {
25          let id = req.params.id;
26          let query = `SELECT subjectName, ratePerHour FROM hm_post A inner join hm_tutor_profile B on
27          (A.tutorProfileId = B.id and B.userId = ?)`;
28          console.log(query);
29          database.query(query, [id], (err, result) => {
30              if (err) console.log(err);
31              else res.json(result);
32          });
33      },
34

```

```

server > controller > JS TutorProfileController.js
33
34      // Review: I think, using handling error with try catch would be better and in case of error, it should return error
35      getTutorQualificationById: async (req, res) => {
36          let id = req.params.id;
37          let query = `SELECT A.id, A.subjectName, A.description, A.grade FROM hm_qualification A
38          inner join hm_tutor_profile B on (A.tutorProfileId = B.id and B.userId = ?)`;
39
40          database.query(query, [id], (err, result) => {
41              if (err) console.log(err);
42              else res.json(result);
43          });
44      },
45
46      // Review: I think, It's preferable to put all code that can cause errors inside try block for efficient error handling.
47      //          // I think, for inner joining query proper using meaningful variable name be better for code readability.
48      getReviewsById: async (req, res) => {
49          let id = req.params.id;
50          try {
51              let result = await executeQuery(
52                  `SELECT A.id, A.text, A.rating, A.createdDateTime, A.modifiedDateTime, U.firstName, U.lastName, A.userId FROM hm_review A
53                  inner join hm_user U on (A.userId = U.id)
54                  inner join hm_tutor_profile T on (A.tutorProfileId = T.id and T.userId = ?)`;
55                  [id]
56              );
57
58              res.status(200).json(result);
59          } catch (error) {
60              res.status(500).json({ message: error });
61          }
62      },
63

```

```
server > controller > JS TutorProfileController.js
173
174 // Review: 1. In case of error handling, I think, we should use try catch functionality
175 // 2. For Picture path, I think, instead of typing image path, using global variable in common file
176 // would be better for minimizing typing error issue.
177 saveTutorInfo: async (req, res) => {
178   await uploadFile(req, res);
179   if (req.file == undefined) {
180     return res.status(400).send({ message: "Please upload a Image!" });
181   }
182
183   let { UserId, About, Age } = req.body;
184   let PicturePath = "public/images/" + req.file.originalname;
185   database.query(
186     "UPDATE hm_tutor_profile SET about = ?, age = ?, rating = 0, picPath = ?, status = 100 WHERE userId = ?",
187     [About, Age, PicturePath, UserId],
188     (err) => {
189       if (err) res.status(400).send(`Response Error: ${err}`);
190       else res.status(200).json({ message: "Tutor profile updated" });
191     }
192   );
193 },
194
195 // Review: 1. In case of error handling, I think, we should use try catch functionality
196 updateTutorInfo: async (req, res) => {
197   const errors = validationResult(req);
198   if (!errors.isEmpty()) {
199     return res.status(400).json({ errors: errors.array() });
200   }
201
202   let { UserId, Status } = req.body;
203   database.query(
204     "UPDATE hm_tutor_profile SET status = ? WHERE userId = ?",
205     [Status, UserId],
206     (err) => {
207       if (err) {
208         res.status(500).json({ message: error });
209       } else {
210         res.json({ message: "Tutor Profile Updated" });
211       }
212     }
213   );
214 },
215
216 // Review: 1. In case of error handling, I think, we should use try catch functionality
217 deleteTutorInfo: async (req, res) => {
218   const errors = validationResult(req);
219   if (!errors.isEmpty()) {
220     return res.status(400).json({ errors: errors.array() });
221   }
222
223   let { UserId } = req.body;
224   database.query(
225     "DELETE FROM hm_tutor_profile WHERE userId = ?",
226     [UserId],
227     (err) => {
228       if (err) {
229         res.status(500).json({ message: error });
230       } else {
231         res.json({ message: "Tutor Profile Deleted" });
232       }
233     }
234   );
235 },
236
237 // Review: 1. In case of error handling, I think, we should use try catch functionality
238 getAllTutorInfo: async (req, res) => {
239   database.query(
240     "SELECT * FROM hm_tutor_profile",
241     (err, result) => {
242       if (err) {
243         res.status(500).json({ message: error });
244       } else {
245         res.json(result);
246       }
247     }
248   );
249 },
250
251 // Review: 1. In case of error handling, I think, we should use try catch functionality
252 getTutorInfoById: async (req, res) => {
253   const errors = validationResult(req);
254   if (!errors.isEmpty()) {
255     return res.status(400).json({ errors: errors.array() });
256   }
257
258   let { UserId } = req.query;
259   database.query(
260     "SELECT * FROM hm_tutor_profile WHERE userId = ?",
261     [UserId],
262     (err, result) => {
263       if (err) {
264         res.status(500).json({ message: error });
265       } else {
266         res.json(result);
267       }
268     }
269   );
270 },
271
272 // Review: 1. In case of error handling, I think, we should use try catch functionality
273 getTutorInfoByRating: async (req, res) => {
274   const errors = validationResult(req);
275   if (!errors.isEmpty()) {
276     return res.status(400).json({ errors: errors.array() });
277   }
278
279   let { Rating } = req.query;
280   database.query(
281     "SELECT * FROM hm_tutor_profile WHERE rating = ?",
282     [Rating],
283     (err, result) => {
284       if (err) {
285         res.status(500).json({ message: error });
286       } else {
287         res.json(result);
288       }
289     }
290   );
291 },
292
293 // Review: 1. In case of error handling, I think, we should use try catch functionality
294 getTutorInfoByStatus: async (req, res) => {
295   const errors = validationResult(req);
296   if (!errors.isEmpty()) {
297     return res.status(400).json({ errors: errors.array() });
298   }
299
300   let { Status } = req.query;
301   database.query(
302     "SELECT * FROM hm_tutor_profile WHERE status = ?",
303     [Status],
304     (err, result) => {
305       if (err) {
306         res.status(500).json({ message: error });
307       } else {
308         res.json(result);
309       }
310     }
311   );
312 },
313
314 // Review: 1. In case of error handling, I think, we should use try catch functionality
315 getTutorInfoByAge: async (req, res) => {
316   const errors = validationResult(req);
317   if (!errors.isEmpty()) {
318     return res.status(400).json({ errors: errors.array() });
319   }
320
321   let { Age } = req.query;
322   database.query(
323     "SELECT * FROM hm_tutor_profile WHERE age = ?",
324     [Age],
325     (err, result) => {
326       if (err) {
327         res.status(500).json({ message: error });
328       } else {
329         res.json(result);
330       }
331     }
332   );
333 },
334
335 // Review: 1. In case of error handling, I think, we should use try catch functionality
336 getTutorInfoByAbout: async (req, res) => {
337   const errors = validationResult(req);
338   if (!errors.isEmpty()) {
339     return res.status(400).json({ errors: errors.array() });
340   }
341
342   let { About } = req.query;
343   database.query(
344     "SELECT * FROM hm_tutor_profile WHERE about = ?",
345     [About],
346     (err, result) => {
347       if (err) {
348         res.status(500).json({ message: error });
349       } else {
350         res.json(result);
351       }
352     }
353   );
354 },
355
356 // Review: 1. In case of error handling, I think, we should use try catch functionality
357 getTutorInfoByPicPath: async (req, res) => {
358   const errors = validationResult(req);
359   if (!errors.isEmpty()) {
360     return res.status(400).json({ errors: errors.array() });
361   }
362
363   let { PicPath } = req.query;
364   database.query(
365     "SELECT * FROM hm_tutor_profile WHERE picPath = ?",
366     [PicPath],
367     (err, result) => {
368       if (err) {
369         res.status(500).json({ message: error });
370       } else {
371         res.json(result);
372       }
373     }
374   );
375 },
376
377 // Review: 1. In case of error handling, I think, we should use try catch functionality
378 getTutorInfoByRatingAndStatus: async (req, res) => {
379   const errors = validationResult(req);
380   if (!errors.isEmpty()) {
381     return res.status(400).json({ errors: errors.array() });
382   }
383
384   let { Rating, Status } = req.query;
385   database.query(
386     "SELECT * FROM hm_tutor_profile WHERE rating = ? AND status = ?",
387     [Rating, Status],
388     (err, result) => {
389       if (err) {
390         res.status(500).json({ message: error });
391       } else {
392         res.json(result);
393       }
394     }
395   );
396 },
397
398 // Review: 1. In case of error handling, I think, we should use try catch functionality
399 getTutorInfoByRatingAndAge: async (req, res) => {
400   const errors = validationResult(req);
401   if (!errors.isEmpty()) {
402     return res.status(400).json({ errors: errors.array() });
403   }
404
405   let { Rating, Age } = req.query;
406   database.query(
407     "SELECT * FROM hm_tutor_profile WHERE rating = ? AND age = ?",
408     [Rating, Age],
409     (err, result) => {
410       if (err) {
411         res.status(500).json({ message: error });
412       } else {
413         res.json(result);
414       }
415     }
416   );
417 },
418
419 // Review: 1. In case of error handling, I think, we should use try catch functionality
420 getTutorInfoByRatingAndAbout: async (req, res) => {
421   const errors = validationResult(req);
422   if (!errors.isEmpty()) {
423     return res.status(400).json({ errors: errors.array() });
424   }
425
426   let { Rating, About } = req.query;
427   database.query(
428     "SELECT * FROM hm_tutor_profile WHERE rating = ? AND about = ?",
429     [Rating, About],
430     (err, result) => {
431       if (err) {
432         res.status(500).json({ message: error });
433       } else {
434         res.json(result);
435       }
436     }
437   );
438 },
439
440 // Review: 1. In case of error handling, I think, we should use try catch functionality
441 getTutorInfoByRatingAndPicPath: async (req, res) => {
442   const errors = validationResult(req);
443   if (!errors.isEmpty()) {
444     return res.status(400).json({ errors: errors.array() });
445   }
446
447   let { Rating, PicPath } = req.query;
448   database.query(
449     "SELECT * FROM hm_tutor_profile WHERE rating = ? AND picPath = ?",
450     [Rating, PicPath],
451     (err, result) => {
452       if (err) {
453         res.status(500).json({ message: error });
454       } else {
455         res.json(result);
456       }
457     }
458   );
459 },
460
461 // Review: 1. In case of error handling, I think, we should use try catch functionality
462 getTutorInfoByStatusAndAge: async (req, res) => {
463   const errors = validationResult(req);
464   if (!errors.isEmpty()) {
465     return res.status(400).json({ errors: errors.array() });
466   }
467
468   let { Status, Age } = req.query;
469   database.query(
470     "SELECT * FROM hm_tutor_profile WHERE status = ? AND age = ?",
471     [Status, Age],
472     (err, result) => {
473       if (err) {
474         res.status(500).json({ message: error });
475       } else {
476         res.json(result);
477       }
478     }
479   );
480 },
481
482 // Review: 1. In case of error handling, I think, we should use try catch functionality
483 getTutorInfoByStatusAndAbout: async (req, res) => {
484   const errors = validationResult(req);
485   if (!errors.isEmpty()) {
486     return res.status(400).json({ errors: errors.array() });
487   }
488
489   let { Status, About } = req.query;
490   database.query(
491     "SELECT * FROM hm_tutor_profile WHERE status = ? AND about = ?",
492     [Status, About],
493     (err, result) => {
494       if (err) {
495         res.status(500).json({ message: error });
496       } else {
497         res.json(result);
498       }
499     }
500   );
501 },
502
503 // Review: 1. In case of error handling, I think, we should use try catch functionality
504 getTutorInfoByStatusAndPicPath: async (req, res) => {
505   const errors = validationResult(req);
506   if (!errors.isEmpty()) {
507     return res.status(400).json({ errors: errors.array() });
508   }
509
510   let { Status, PicPath } = req.query;
511   database.query(
512     "SELECT * FROM hm_tutor_profile WHERE status = ? AND picPath = ?",
513     [Status, PicPath],
514     (err, result) => {
515       if (err) {
516         res.status(500).json({ message: error });
517       } else {
518         res.json(result);
519       }
520     }
521   );
522 },
523
524 // Review: 1. In case of error handling, I think, we should use try catch functionality
525 getTutorInfoByAgeAndAbout: async (req, res) => {
526   const errors = validationResult(req);
527   if (!errors.isEmpty()) {
528     return res.status(400).json({ errors: errors.array() });
529   }
530
531   let { Age, About } = req.query;
532   database.query(
533     "SELECT * FROM hm_tutor_profile WHERE age = ? AND about = ?",
534     [Age, About],
535     (err, result) => {
536       if (err) {
537         res.status(500).json({ message: error });
538       } else {
539         res.json(result);
540       }
541     }
542   );
543 },
544
545 // Review: 1. In case of error handling, I think, we should use try catch functionality
546 getTutorInfoByAgeAndPicPath: async (req, res) => {
547   const errors = validationResult(req);
548   if (!errors.isEmpty()) {
549     return res.status(400).json({ errors: errors.array() });
550   }
551
552   let { Age, PicPath } = req.query;
553   database.query(
554     "SELECT * FROM hm_tutor_profile WHERE age = ? AND picPath = ?",
555     [Age, PicPath],
556     (err, result) => {
557       if (err) {
558         res.status(500).json({ message: error });
559       } else {
560         res.json(result);
561       }
562     }
563   );
564 },
565
566 // Review: 1. In case of error handling, I think, we should use try catch functionality
567 getTutorInfoByAboutAndPicPath: async (req, res) => {
568   const errors = validationResult(req);
569   if (!errors.isEmpty()) {
570     return res.status(400).json({ errors: errors.array() });
571   }
572
573   let { About, PicPath } = req.query;
574   database.query(
575     "SELECT * FROM hm_tutor_profile WHERE about = ? AND picPath = ?",
576     [About, PicPath],
577     (err, result) => {
578       if (err) {
579         res.status(500).json({ message: error });
580       } else {
581         res.json(result);
582       }
583     }
584   );
585 },
586
587 // Review: 1. In case of error handling, I think, we should use try catch functionality
588 getTutorInfoByRatingAndStatusAndAge: async (req, res) => {
589   const errors = validationResult(req);
590   if (!errors.isEmpty()) {
591     return res.status(400).json({ errors: errors.array() });
592   }
593
594   let { Rating, Status, Age } = req.query;
595   database.query(
596     "SELECT * FROM hm_tutor_profile WHERE rating = ? AND status = ? AND age = ?",
597     [Rating, Status, Age],
598     (err, result) => {
599       if (err) {
600         res.status(500).json({ message: error });
601       } else {
602         res.json(result);
603       }
604     }
605   );
606 },
607
608 // Review: 1. In case of error handling, I think, we should use try catch functionality
609 getTutorInfoByRatingAndStatusAndAbout: async (req, res) => {
610   const errors = validationResult(req);
611   if (!errors.isEmpty()) {
612     return res.status(400).json({ errors: errors.array() });
613   }
614
615   let { Rating, Status, About } = req.query;
616   database.query(
617     "SELECT * FROM hm_tutor_profile WHERE rating = ? AND status = ? AND about = ?",
618     [Rating, Status, About],
619     (err, result) => {
620       if (err) {
621         res.status(500).json({ message: error });
622       } else {
623         res.json(result);
624       }
625     }
626   );
627 },
628
629 // Review: 1. In case of error handling, I think, we should use try catch functionality
630 getTutorInfoByRatingAndStatusAndPicPath: async (req, res) => {
631   const errors = validationResult(req);
632   if (!errors.isEmpty()) {
633     return res.status(400).json({ errors: errors.array() });
634   }
635
636   let { Rating, Status, PicPath } = req.query;
637   database.query(
638     "SELECT * FROM hm_tutor_profile WHERE rating = ? AND status = ? AND picPath = ?",
639     [Rating, Status, PicPath],
640     (err, result) => {
641       if (err) {
642         res.status(500).json({ message: error });
643       } else {
644         res.json(result);
645       }
646     }
647   );
648 },
649
650 // Review: 1. In case of error handling, I think, we should use try catch functionality
651 getTutorInfoByAgeAndAboutAndPicPath: async (req, res) => {
652   const errors = validationResult(req);
653   if (!errors.isEmpty()) {
654     return res.status(400).json({ errors: errors.array() });
655   }
656
657   let { Age, About, PicPath } = req.query;
658   database.query(
659     "SELECT * FROM hm_tutor_profile WHERE age = ? AND about = ? AND picPath = ?",
660     [Age, About, PicPath],
661     (err, result) => {
662       if (err) {
663         res.status(500).json({ message: error });
664       } else {
665         res.json(result);
666       }
667     }
668   );
669 },
670
671 // Review: 1. In case of error handling, I think, we should use try catch functionality
672 getTutorInfoByStatusAndAgeAndAbout: async (req, res) => {
673   const errors = validationResult(req);
674   if (!errors.isEmpty()) {
675     return res.status(400).json({ errors: errors.array() });
676   }
677
678   let { Status, Age, About } = req.query;
679   database.query(
680     "SELECT * FROM hm_tutor_profile WHERE status = ? AND age = ? AND about = ?",
681     [Status, Age, About],
682     (err, result) => {
683       if (err) {
684         res.status(500).json({ message: error });
685       } else {
686         res.json(result);
687       }
688     }
689   );
690 },
691
692 // Review: 1. In case of error handling, I think, we should use try catch functionality
693 getTutorInfoByStatusAndAgeAndPicPath: async (req, res) => {
694   const errors = validationResult(req);
695   if (!errors.isEmpty()) {
696     return res.status(400).json({ errors: errors.array() });
697   }
698
699   let { Status, Age, PicPath } = req.query;
700   database.query(
701     "SELECT * FROM hm_tutor_profile WHERE status = ? AND age = ? AND picPath = ?",
702     [Status, Age, PicPath],
703     (err, result) => {
704       if (err) {
705         res.status(500).json({ message: error });
706       } else {
707         res.json(result);
708       }
709     }
710   );
711 },
712
713 // Review: 1. In case of error handling, I think, we should use try catch functionality
714 getTutorInfoByAboutAndAgeAndPicPath: async (req, res) => {
715   const errors = validationResult(req);
716   if (!errors.isEmpty()) {
717     return res.status(400).json({ errors: errors.array() });
718   }
719
720   let { About, Age, PicPath } = req.query;
721   database.query(
722     "SELECT * FROM hm_tutor_profile WHERE about = ? AND age = ? AND picPath = ?",
723     [About, Age, PicPath],
724     (err, result) => {
725       if (err) {
726         res.status(500).json({ message: error });
727       } else {
728         res.json(result);
729       }
730     }
731   );
732 },
733
734 // Review: 1. In case of error handling, I think, we should use try catch functionality
735 getTutorInfoByRatingAndStatusAndAgeAndAbout: async (req, res) => {
736   const errors = validationResult(req);
737   if (!errors.isEmpty()) {
738     return res.status(400).json({ errors: errors.array() });
739   }
740
741   let { Rating, Status, Age, About } = req.query;
742   database.query(
743     "SELECT * FROM hm_tutor_profile WHERE rating = ? AND status = ? AND age = ? AND about = ?",
744     [Rating, Status, Age, About],
745     (err, result) => {
746       if (err) {
747         res.status(500).json({ message: error });
748       } else {
749         res.json(result);
750       }
751     }
752   );
753 },
754
755 // Review: 1. In case of error handling, I think, we should use try catch functionality
756 getTutorInfoByRatingAndStatusAndAgeAndPicPath: async (req, res) => {
757   const errors = validationResult(req);
758   if (!errors.isEmpty()) {
759     return res.status(400).json({ errors: errors.array() });
760   }
761
762   let { Rating, Status, Age, PicPath } = req.query;
763   database.query(
764     "SELECT * FROM hm_tutor_profile WHERE rating = ? AND status = ? AND age = ? AND picPath = ?",
765     [Rating, Status, Age, PicPath],
766     (err, result) => {
767       if (err) {
768         res.status(500).json({ message: error });
769       } else {
770         res.json(result);
771       }
772     }
773   );
774 },
775
776 // Review: 1. In case of error handling, I think, we should use try catch functionality
777 getTutorInfoByAgeAndAboutAndAgeAndPicPath: async (req, res) => {
778   const errors = validationResult(req);
779   if (!errors.isEmpty()) {
780     return res.status(400).json({ errors: errors.array() });
781   }
782
783   let { Age, About, Age, PicPath } = req.query;
784   database.query(
785     "SELECT * FROM hm_tutor_profile WHERE age = ? AND about = ? AND age = ? AND picPath = ?",
786     [Age, About, Age, PicPath],
787     (err, result) => {
788       if (err) {
789         res.status(500).json({ message: error });
790       } else {
791         res.json(result);
792       }
793     }
794   );
795 },
796
797 // Review: 1. In case of error handling, I think, we should use try catch functionality
798 getTutorInfoByStatusAndAgeAndAboutAndPicPath: async (req, res) => {
799   const errors = validationResult(req);
800   if (!errors.isEmpty()) {
801     return res.status(400).json({ errors: errors.array() });
802   }
803
804   let { Status, Age, About, PicPath } = req.query;
805   database.query(
806     "SELECT * FROM hm_tutor_profile WHERE status = ? AND age = ? AND about = ? AND picPath = ?",
807     [Status, Age, About, PicPath],
808     (err, result) => {
809       if (err) {
810         res.status(500).json({ message: error });
811       } else {
812         res.json(result);
813       }
814     }
815   );
816 },
817
818 // Review: 1. In case of error handling, I think, we should use try catch functionality
819 getTutorInfoByAboutAndAgeAndAboutAndPicPath: async (req, res) => {
820   const errors = validationResult(req);
821   if (!errors.isEmpty()) {
822     return res.status(400).json({ errors: errors.array() });
823   }
824
825   let { About, Age, About, PicPath } = req.query;
826   database.query(
827     "SELECT * FROM hm_tutor_profile WHERE about = ? AND age = ? AND about = ? AND picPath = ?",
828     [About, Age, About, PicPath],
829     (err, result) => {
830       if (err) {
831         res.status(500).json({ message: error });
832       } else {
833         res.json(result);
834       }
835     }
836   );
837 },
838
839 // Review: 1. In case of error handling, I think, we should use try catch functionality
840 getTutorInfoByRatingAndStatusAndAgeAndAboutAndPicPath: async (req, res) => {
841   const errors = validationResult(req);
842   if (!errors.isEmpty()) {
843     return res.status(400).json({ errors: errors.array() });
844   }
845
846   let { Rating, Status, Age, About, PicPath } = req.query;
847   database.query(
848     "SELECT * FROM hm_tutor_profile WHERE rating = ? AND status = ? AND age = ? AND about = ? AND picPath = ?",
849     [Rating, Status, Age, About, PicPath],
850     (err, result) => {
851       if (err) {
852         res.status(500).json({ message: error });
853       } else {
854         res.json(result);
855       }
856     }
857   );
858 },
859
860 // Review: 1. In case of error handling, I think, we should use try catch functionality
861 getTutorInfoByAgeAndAboutAndAgeAndAboutAndPicPath: async (req, res) => {
862   const errors = validationResult(req);
863   if (!errors.isEmpty()) {
864     return res.status(400).json({ errors: errors.array() });
865   }
866
867   let { Age, About, Age, About, PicPath } = req.query;
868   database.query(
869     "SELECT * FROM hm_tutor_profile WHERE age = ? AND about = ? AND age = ? AND about = ? AND picPath = ?",
870     [Age, About, Age, About, PicPath],
871     (err, result) => {
872       if (err) {
873         res.status(500).json({ message: error });
874       } else {
875         res.json(result);
876       }
877     }
878   );
879 },
880
881 // Review: 1. In case of error handling, I think, we should use try catch functionality
882 getTutorInfoByStatusAndAgeAndAboutAndAgeAndPicPath: async (req, res) => {
883   const errors = validationResult(req);
884   if (!errors.isEmpty()) {
885     return res.status(400).json({ errors: errors.array() });
886   }
887
888   let { Status, Age, About, Age, PicPath } = req.query;
889   database.query(
890     "SELECT * FROM hm_tutor_profile WHERE status = ? AND age = ? AND about = ? AND age = ? AND picPath = ?",
891     [Status, Age, About, Age, PicPath],
892     (err, result) => {
893       if (err) {
894         res.status(500).json({ message: error });
895       } else {
896         res.json(result);
897       }
898     }
899   );
900 },
901
902 // Review: 1. In case of error handling, I think, we should use try catch functionality
903 getTutorInfoByAboutAndAgeAndAboutAndAgeAndPicPath: async (req, res) => {
904   const errors = validationResult(req);
905   if (!errors.isEmpty()) {
906     return res.status(400).json({ errors: errors.array() });
907   }
908
909   let { About, Age, About, Age, PicPath } = req.query;
910   database.query(
911     "SELECT * FROM hm_tutor_profile WHERE about = ? AND age = ? AND about = ? AND age = ? AND picPath = ?",
912     [About, Age, About, Age, PicPath],
913     (err, result) => {
914       if (err) {
915         res.status(500).json({ message: error });
916       } else {
917         res.json(result);
918       }
919     }
920   );
921 },
922
923 // Review: 1. In case of error handling, I think, we should use try catch functionality
924 getTutorInfoByRatingAndStatusAndAgeAndAboutAndAgeAndPicPath: async (req, res) => {
925   const errors = validationResult(req);
926   if (!errors.isEmpty()) {
927     return res.status(400).json({ errors: errors.array() });
928   }
929
930   let { Rating, Status, Age, About, Age, PicPath } = req.query;
931   database.query(
932     "SELECT * FROM hm_tutor_profile WHERE rating = ? AND status = ? AND age = ? AND about = ? AND age = ? AND picPath = ?",
933     [Rating, Status, Age, About, Age, PicPath],
934     (err, result) => {
935       if (err) {
936         res.status(500).json({ message: error });
937       } else {
938         res.json(result);
939       }
940     }
941   );
942 },
943
944 // Review: 1. In case of error handling, I think, we should use try catch functionality
945 getTutorInfoByAgeAndAboutAndAgeAndAboutAndAgeAndPicPath: async (req, res) => {
946   const errors = validationResult(req);
947   if (!errors.isEmpty()) {
948     return res.status(400).json({ errors: errors.array() });
949   }
950
951   let { Age, About, Age, About, Age, PicPath } = req.query;
952   database.query(
953     "SELECT * FROM hm_tutor_profile WHERE age = ? AND about = ? AND age = ? AND about = ? AND age = ? AND picPath = ?",
954     [Age, About, Age, About, Age, PicPath],
955     (err, result) => {
956       if (err) {
957         res.status(500).json({ message: error });
958       } else {
959         res.json(result);
960       }
961     }
962   );
963 },
964
965 // Review: 1. In case of error handling, I think, we should use try catch functionality
966 getTutorInfoByStatusAndAgeAndAboutAndAgeAndAboutAndPicPath: async (req, res) => {
967   const errors = validationResult(req);
968   if (!errors.isEmpty()) {
969     return res.status(400).json({ errors: errors.array() });
970   }
971
972   let { Status, Age, About, Age, About, PicPath } = req.query;
973   database.query(
974     "SELECT * FROM hm_tutor_profile WHERE status = ? AND age = ? AND about = ? AND age = ? AND about = ? AND picPath = ?",
975     [Status, Age, About, Age, About, PicPath],
976     (err, result) => {
977       if (err) {
978         res.status(500).json({ message: error });
979       } else {
980         res.json(result);
981       }
982     }
983   );
984 },
985
986 // Review: 1. In case of error handling, I think, we should use try catch functionality
987 getTutorInfoByAboutAndAgeAndAboutAndAgeAndAboutAndPicPath: async (req, res) => {
988   const errors = validationResult(req);
989   if (!errors.isEmpty()) {
990     return res.status(400).json({ errors: errors.array() });
991   }
992
993   let { About, Age, About, Age, About, PicPath } = req.query;
994   database.query(
995     "SELECT * FROM hm_tutor_profile WHERE about = ? AND age = ? AND about = ? AND age = ? AND about = ? AND picPath = ?",
996     [About, Age, About, Age, About, PicPath],
997     (err, result) => {
998       if (err) {
999         res.status(500).json({ message: error });
1000       } else {
1001         res.json(result);
1002       }
1003     }
1004   );
1005 },
1006
1007 // Review: 1. In case of error handling, I think, we should use try catch functionality
1008 getTutorInfoByRatingAndStatusAndAgeAndAboutAndAgeAndAboutAndPicPath: async (req, res) => {
1009   const errors = validationResult(req);
1010   if (!errors.isEmpty()) {
1011     return res.status(400).json({ errors: errors.array() });
1012   }
1013
1014   let { Rating, Status, Age, About, Age, About, PicPath } = req.query;
1015   database.query(
1016     "SELECT * FROM hm_tutor_profile WHERE rating = ? AND status = ? AND age = ? AND about = ? AND age = ? AND about = ? AND picPath = ?",
1017     [Rating, Status, Age, About, Age, About, PicPath],
1018     (err, result) => {
1019       if (err) {
1020         res.status(500).json({ message: error });
1021       } else {
1022         res.json(result);
1023       }
1024     }
1025   );
1026 },
1027
1028 // Review: 1. In case of error handling, I think, we should use try catch functionality
1029 getTutorInfoByAgeAndAboutAndAgeAndAboutAndAgeAndAboutAndPicPath: async (req, res) => {
1030   const errors = validationResult(req);
1031   if (!errors.isEmpty()) {
1032     return res.status(400).json({ errors: errors.array() });
1033   }
1034
1035   let { Age, About, Age, About, Age, About, PicPath } = req.query;
1036   database.query(
1037     "SELECT * FROM hm_tutor_profile WHERE age = ? AND about = ? AND age = ? AND about = ? AND age = ? AND about = ? AND picPath = ?",
1038     [Age, About, Age, About, Age, About, PicPath],
1039     (err, result) => {
1040       if (err) {
1041         res.status(500).json({ message: error });
1042       } else {
1043         res.json(result);
1044       }
1045     }
1046   );
1047 },
1048
1049 // Review: 1. In case of error handling, I think, we should use try catch functionality
1050 getTutorInfoByStatusAndAgeAndAboutAndAgeAndAboutAndAgeAndPicPath: async (req, res) => {
1051   const errors = validationResult(req);
1052   if (!errors.isEmpty()) {
1053     return res.status(400).json({ errors: errors.array() });
1054   }
1055
1056   let { Status, Age, About, Age, About, Age, PicPath } = req.query;
1057   database.query(
1058     "SELECT * FROM hm_tutor_profile WHERE status = ? AND age = ? AND about = ? AND age = ? AND about = ? AND age = ? AND picPath = ?",
1059     [Status, Age, About, Age, About, Age, PicPath],
1060     (err, result) => {
1061       if (err) {
1062         res.status(500).json({ message: error });
1063       } else {
1064         res.json(result);
1065       }
1066     }
1067   );
1068 },
1069
1070 // Review: 1. In case of error handling, I think, we should use try catch functionality
1071 getTutorInfoByAboutAndAgeAndAboutAndAgeAndAboutAndAgeAndPicPath: async (req, res) => {
1072   const errors = validationResult(req);
1073   if (!errors.isEmpty()) {
1074     return res.status(400).json({ errors: errors.array() });
1075   }
1076
1077   let { About, Age, About, Age, About, Age, PicPath } = req.query;
1078   database.query(
1079     "SELECT * FROM hm_tutor_profile WHERE about = ? AND age = ? AND about = ? AND age = ? AND about = ? AND age = ? AND picPath = ?",
1080     [About, Age, About, Age, About, Age, PicPath],
1081     (err, result) => {
1082       if (err) {
1083         res.status(500).json({ message: error });
1084       } else {
1085         res.json(result);
1086       }
1087     }
1088   );
1089 },
1090
1091 // Review: 1. In case of error handling, I think, we should use try catch functionality
1092 getTutorInfoByRatingAndStatusAndAgeAndAboutAndAgeAndAboutAndAgeAndPicPath: async (req, res) => {
1093   const errors = validationResult(req);
1094   if (!errors.isEmpty()) {
1095     return res.status(400).json({ errors: errors.array() });
1096   }
1097
1098   let { Rating, Status, Age, About, Age, About, Age, PicPath } = req.query;
1099   database.query(
1100     "SELECT * FROM hm_tutor_profile WHERE rating = ? AND status = ? AND age = ? AND about = ? AND age = ? AND about = ? AND age = ? AND picPath = ?",
1101     [Rating, Status, Age, About, Age, About, Age, PicPath],
1102     (err, result) => {
1103       if (err) {
1104         res.status(500).json({ message: error });
1105       } else {
1106         res.json(result);
1107       }
1108     }
1109   );
1110 },
1111
1112 // Review: 1. In case of error handling, I think, we should use try catch functionality
1113 getTutorInfoByAgeAndAboutAndAgeAndAboutAndAgeAndAboutAndAgeAndPicPath: async (req, res) => {
1114   const errors = validationResult(req);
1115   if (!errors.isEmpty()) {
1116     return res.status(400).json({ errors: errors.array() });
1117   }
1118
1119   let { Age, About, Age, About, Age, About, Age, PicPath } = req.query;
1120   database.query(
1121     "SELECT * FROM hm_tutor_profile WHERE age = ? AND about = ? AND age = ? AND about = ? AND age = ? AND about = ? AND age = ? AND picPath = ?",
1122     [Age, About, Age, About, Age, About, Age, PicPath],
1123     (err, result) => {
1124       if (err) {
1125         res.status(500).json({ message: error });
1126       } else {
1127         res.json(result);
1128       }
1129     }
1130   );
1131 },
1132
1133 // Review: 1. In case of error handling, I think, we should use try catch functionality
1134 getTutorInfoByStatusAndAgeAndAboutAndAgeAndAboutAndAgeAndAgeAndPicPath: async (req, res) => {
1135   const errors = validationResult(req);
1136   if (!errors.isEmpty()) {
1137     return res.status(400).json({ errors: errors.array() });
1138   }
1139
1140   let { Status, Age, About, Age, About, Age, Age, PicPath } = req.query;
1141   database.query(
1142     "SELECT * FROM hm_tutor_profile WHERE status = ? AND age = ? AND about = ? AND age = ? AND about = ? AND age = ? AND age = ? AND picPath = ?",
1143     [Status, Age, About, Age, About, Age, Age, PicPath],
1144     (err, result) => {
1145       if (err) {
1146         res.status(500).json({ message: error });
1147       } else {
1148         res.json(result);
1149       }
1150     }
1151   );
1
```

```

client > src > pages > tutorProfile > reviewList > JS ReviewList.js
1 import React, { useState, useEffect, useRef } from "react";
2 import { useSelector, useDispatch } from "react-redux";
3 import { useParams } from "react-router-dom";
4 import moment from "moment";
5 import { getTutorReviewDataById } from "../../core/selectors/tutor";
6 import { getTutorReviewById } from "../../core/actionCreators/tutor";
7 import { setTutorReview } from "../../core/actionCreators/tutor";
8 import { saveReview } from "../../core/actionCreators/tutor";
9 import { ListGroup, Row, Col, Button, Form } from "react-bootstrap";
10 import Rate from "rc-rate";
11 import "rc-rate/assets/index.css";
12 import { getCurrentUser, getUserType } from "../../core/selectors/user";
13
14 // Destructuring props would be a good idea instead of using whole props
15 export default function ReviewList(props) {
16   const dispatch = useDispatch();
17
18   let starCountRef = useRef(null);
19   const textReviewRef = useRef(null);
20   const user = useSelector(getCurrentUser);
21   const userType = useSelector(getUserType);
22
23   // As we are using react router so, using useParams hook to get tutorId would be cleaner to do that.
24
25   let { tutorId } = useParams();
26   if (props.tutorId !== undefined && props.tutorId !== "") {
27     tutorId = props.tutorId;
28   }
29   const tutorReviewData = useSelector(getTutorReviewDataById);
30   const [tutorReviews, setTutorReview] = useState([]);
31
32   useEffect(() => {
33     dispatch(getTutorReviewById(tutorId));
34   }, []);
35   useEffect(() => {
36     setTutorReview(tutorReviewData);
37   }, [tutorReviewData]);
38

```

```

client > src > pages > tutorProfile > reviewList > JS ReviewList.js
39
40 const submitReview = () => {
41   let review = {
42     Rating: starCountRef.current.state.value,
43     Text: textReviewRef.current.value,
44     UserId: user.id,
45     TutorProfileId: Number(tutorId),
46   };
47   dispatch(saveReview(review));
48 };
49
50 const renderReview = () => {
51   if (userType !== "student") return null;
52
53   return (
54     <div>
55       <Row>
56         <span>YOUR REVIEW</span>
57         <Rate
58           // May be using global variable for default value instead of hard coding default value? You, seconds
59           defaultValue={2.5}
60           ref={starCountRef}
61           allowHalf
62           allowClear={false}
63         />
64         <Col sm={11}>
65           <Form.Control size="md" ref={textReviewRef} type="text" />
66         </Col>
67         <Col sm={1}>
68           <Button
69             className="float-end"
70             variant="primary"
71             size="md"
72             onClick={submitReview}
73             type="submit"
74           >
75             Submit
76         </Button>
77       </Row>
78     </div>
79   );
80 };

```



```

83 const renderReviews = () => {
84   // I think, when we return a Null component we still have a full lifecycle that will trigger depending on what we do on their parent
85   // component. I think, more correct way to do is to do the conditionals on the parent component to avoid even call that child component
86   if ( tutorReviews === undefined || tutorReviews.length === undefined || tutorReviews.length === 0) { return null; }
87   return (
88     <div>
89       <span>REVIEWS</span>
90       /* May be move that to a css class? */
91       <ListGroup style={{ padding: "1.0rem 0 0 0" }}>
92         {tutorReviewData?.map((item, i) => {
93           return (
94             <ListGroup.Item
95               key={i}
96               className="d-flex justify-content-between align-items-start"
97             >
98               <div className="me-auto">
99                 <div className="fw-bold">`${item.firstName} ${item.lastName}`</div>
100                 <div>
101                   <Rate defaultValue={item.rating} disabled />
102                   <span className="text-muted">{item.modifiedDateTime}</span>
103                 </div>
104                 <div className="fw-light">{item.text}</div>
105               </div>
106             </ListGroup.Item>
107           );
108         })}
109       </ListGroup>
110     </div>
111   );
112 };
113 return (
114   <div>
115     {renderReview()}
116     {renderReviews()}
117   </div>
118 );
119 }

```

Self Check on best practices for security

List of major assets that we should protect

- Passwords
- Admin routes
- Private routes

List of major threats for each asset above

- **Passwords:** should be encrypted, otherwise if the database is hacked and the passwords are stored as a plain text, then all the accounts will be exposed.
- **Admin routes:** admin functionalities are critical and only authorized users who have admin privilege can access these routes, otherwise, any user can manipulate the posts and site users improperly.
- **Private routes:** only authenticated users should access these routes like the chatting and adding posts routes.

For each asset, how we may protect it

- **Passwords:** using encryption, so all passwords are hashed and saved in the database.
- **Admin routes:** verifying the token sent in the request header and checking the user role before allowing him to access the API using the "checkAdmin" middleware.
- **Private routes:** verifying the token sent in the request header and checking if it's valid before allowing him to access the API using the "checkAuth" middleware.
- On the client side the admin and private routes are not accessed by not logged in users.

Confirm that you encrypt PW in the DB

We are saving the passwords hashed and not as plain text in the database using the Blowfish Cipher which is one way hashing and cannot be converted to the plain text password.

Confirm Input data validation

- **Valid Email address:** a checking function is used in the signup to verify that the email address has a valid format and is related to Fulda/San Francisco university.
- **Strong Rules for passwords:** a checking function is used in the signup to enforce the user to choose a strong password that complies with the rules of the website (8 min length, one small, one capital one digit and one special char).
- **Limit the search field for up to 40 characters max:** limiting the size of the input field to 40 characters max, and preventing the user to exceed this number.

Self-check: Adherence to original Non-functional specs

List of non functional requirements	Done	On Track
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	✓	
Application shall be optimized for standard desktop/laptop browsers e.g. must render correctly on the two latest versions of two major browsers	✓	
All or selected application functions must render well on mobile devices	✓	
Data shall be stored in the database on the team's deployment cloud server	✓	
No more than 50 concurrent users shall be accessing the application at any time	✓	
Privacy of users shall be protected and all privacy policies will be appropriately communicated to the users.	✓	
The language used shall be English (no localization needed)	✓	
Application shall be very easy to use and intuitive	✓	
Application should follow established architecture patterns		✓
Application code and its repository shall be easy to inspect and maintain	✓	
No email clients shall be allowed.	✓	
Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI.	✓	
Site security: basic best practices shall be applied (as covered in the class) for main data items	✓	
Application shall be media rich (images, video etc.). Media formats shall be standard as used in the market today		✓
Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development	✓	
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	✓	
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.	✓	