ReactJS-NodeJS Learning App

This application is expected to be developed using ReactJS and NodeJS. Kindly find the problem statement and GIF image on this page.

invoice

Note:

- Kindly use **fetch API** for integration.

FRONT-END ReactJS

- This application uses ReactJS as the front-end. Fix the test case errors in the application. The test cases are written in **Jest**.

App Component

- Write a function to get the course details from the backend using the **handleGetData** function.
- Display the details in the card to view the details.

- Write a function to apply for a course using the **handleApp** function.
- Write a function to apply rating a course. The rating can be added only if we have applied for the course.
- Use **isRated** property to disable the add rating option.
- Use **handleRating** to set the rate value in the state.
- Use **handleAddRating** to add rating on clicking Add button.
- Write a function to drop for a course using the **handleDrop** function.

BACK-END: NodeJS

- You will be making use of **Node.js** and **MongoDB** for back-end purposes.
- **learning-app-easy** is the name of the database used in the application.
- There is a single collection called **courses** inside the database learning-app-easy.

COLLECTIONS:

There are is a single file for collection namely courses.js that reside inside Nodejs/src/mongoose/models. The schema for those collections are as follows.

Courses					
Sl. No.	**Field name**	**Type**	**Validations**	**Required**	**Default**
1	_id	ObjectID	Auto-generated	-	-
2	courseName	String	Should not contain any special characters except dots or spaces and length should not be lesser than 3	TRUE	-
3	courseDept	Number	Can only have any one of the following values(WD, AI, DS, CS, CC, UI, GD)	TRUE	-
4	description	String	Should have at least three words	TRUE	-
5	duration	Number	Minimum = 1, Maximum = 100	TRUE	-
6	isApplied	Boolean	-	False	False
7	isRated	Boolean	-	False	False
8	noOfRatings	Number	-	False	0
9	rating	Number	-	False	0
10	__v	Number	Auto-generated	-	0

ROUTERS:

There is a single file namely users.js that contains all the endpoints of the app and resides inside **Nodejs/src/routers**. The endpoints and their functionalities are as follows.

1)**/courses/enroll/:id** -> POST Method -> This route should enrol the user to the course in the courses collection that has _id equal to id that comes with the request by making the isApplied status of the course as true.

```
{
  "message": "You have successfully enrolled for the course"
}
```

```
{
  "error": "You have already applied for this course"
}
```

- If the user is enrolled in the course successfully, then you should send a response code of 200 with the following response message.
- If the user had already enrolled in the course (i.e.) isApplied status of the course is already true, then you should send a response code of 403 with the following response message.

1) *\/courses/enroll/:id* -> POST Method -> This route should enrol the user to the course in the courses collection that has _id equal to id that comes with the request by making the isApplied status of the course as true.

- If the user is enrolled in the course successfully, then you should send a response code of **200** with the following response message.

```
{
  "message": "You have successfully enrolled for the course"
}
```

- If the user had already enrolled in the course (i.e.) isApplied status of the course is already true, then you should send a response code of **403** with the following response message.

```
{
  "error": "You have already applied for this course"
}
```

- If something goes wrong in executing the request, then you should send a status code of **400**.

2) *\/courses/drop/:id* -> DELETE Method -> This route should drop the user from the course in the courses collection that has _id equal to id that comes with the request by making the isApplied status of the course false.

- If the user is dropped from the course successfully, then you should send a response code of **200** with the following response message.

```
{
  "message": "You have dropped the course"
}
```

- If the user hasn't enrolled on the course (i.e.) isApplied status of the course is already false, then you should send a response code of **403** with the following response message.

```
{
  "error": "You have not enrolled for this course"
}
```

- If something goes wrong in executing the request, then you should send a status code of **400**.

3) ***/courses/get*** **-> GET Method ->** This route should fetch all the data from the **courses** collection as the response.

- If the data is fetched successfully, then, you should send a response code of **200**.
- If the data fetching was unsuccessful, then, you should send a response code of **400**.

4) ***/courses/rating/:id*** **-> PATCH Method ->** This route should update the rating and noOfRatings of the course in the **courses** collection that has _id the same as the id which comes with the request URL as follows.

- Increment the **noOfRatings** by 1
- Update the previous rating based on the rating that comes with the request body and the **noOfRatings** .
- Make sure the rating is rounded off to one decimal place.
- Then update the **isRated** property of the course to be true.

Steps to update the rating:

- The previous rating should be multiplied with the previous number noOfRatings.
- The rating that comes with the body should be added to that value.
- The obtained value should be divided with previous noOfRatings + 1.

Sample request: /courses/rating/5ff4264d9a608c280d745c3c

Data sent with the request body:

```
{
  "rating": 4
}
```

This route should have to update the rating and noOfRatings of the course with _id 5ff4264d9a608c280d745c3c.

Sample updated values:

Sl.No.	Rating sent with the request	Values before sending the request			Values after sending the request		
		Rating	noOfRatings	isRated	Rating	noOfRatings	isRated
1	2	4.5	4	false	4	5	true
2	5	3.8	8	false	3.9	9	true
3	3	4.2	10	false	4.1	11	true

- If the rating was updated successfully, then, you should send a response code of **200** with the following response message.

```
{
  "message": "You have rated this course"
}
```

- If the user had already rated the course (i.e.), if the isRated property of the course is already true, then, you should send a response code of **403** with the following response.

```
{
  "error": "You have already rated this course"
}
```

- If the user is not applied for the course and trying to rate the course (i.e.) if the isApplied property of the course is false, then, you should send a response code of **403** with the following message.

```
{
  "error": "You have not enrolled for this course"
}
```

- If something goes wrong in executing the request, then, you should send a response code of **400**.

MongoDB Commands:

- You can open the mongo shell by running **mongo** from the terminal.
- You can view all the data from the database in MongoDB by running **show dbs** from the mongo shell.
- You can select the database by running **use learning-app-easy.**
- You can view the names of the collections by running **show collections**.
- You can view the data inside a collection by running **db.collection_name.find()**
- Press **ctrl+c** to exit.