

## BÀI THỰC HÀNH 3

### HOÀN THIỆN BACK-END CHO ỨNG DỤNG MINH HOẠ

Bài 1: Thiết lập định tuyến cho các thao tác với review trong ứng dụng minh hoạ. Việc này cần thực hiện trong tệp tin `movies.router.js`

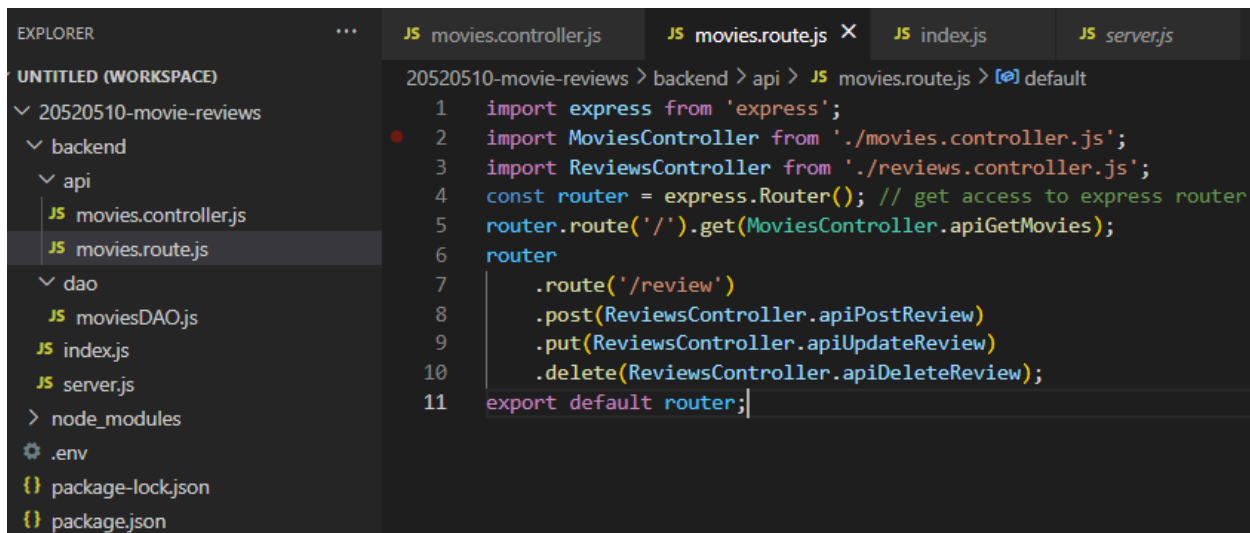
1.1 Định tuyến này sẽ có đường dẫn cuối cùng là `‘/review’`

Ví dụ: `localhost:3000/api/v1/movies/review`

1.2 Thiết lập định tuyến thêm review vào db thông qua phương thức `post`, và gọi đến phương thức `apiPostReview` trong class `ReviewsController` trong tệp tin `reviews.controller.js` mà ta sẽ thiết lập ở bài 2.

1.3 Thiết lập định tuyến sửa review trên db thông qua phương thức `put`, và gọi đến phương thức `apiUpdateReview` trong class `ReviewsController` trong tệp tin `reviews.controller.js` mà ta sẽ thiết lập ở bài 2.

1.4 Thiết lập định tuyến xoá review trên db thông qua phương thức `delete`, và gọi đến phương thức `apiDeleteReview` trong class `ReviewsController` trong tệp tin `reviews.controller.js` mà ta sẽ thiết lập ở bài 2.

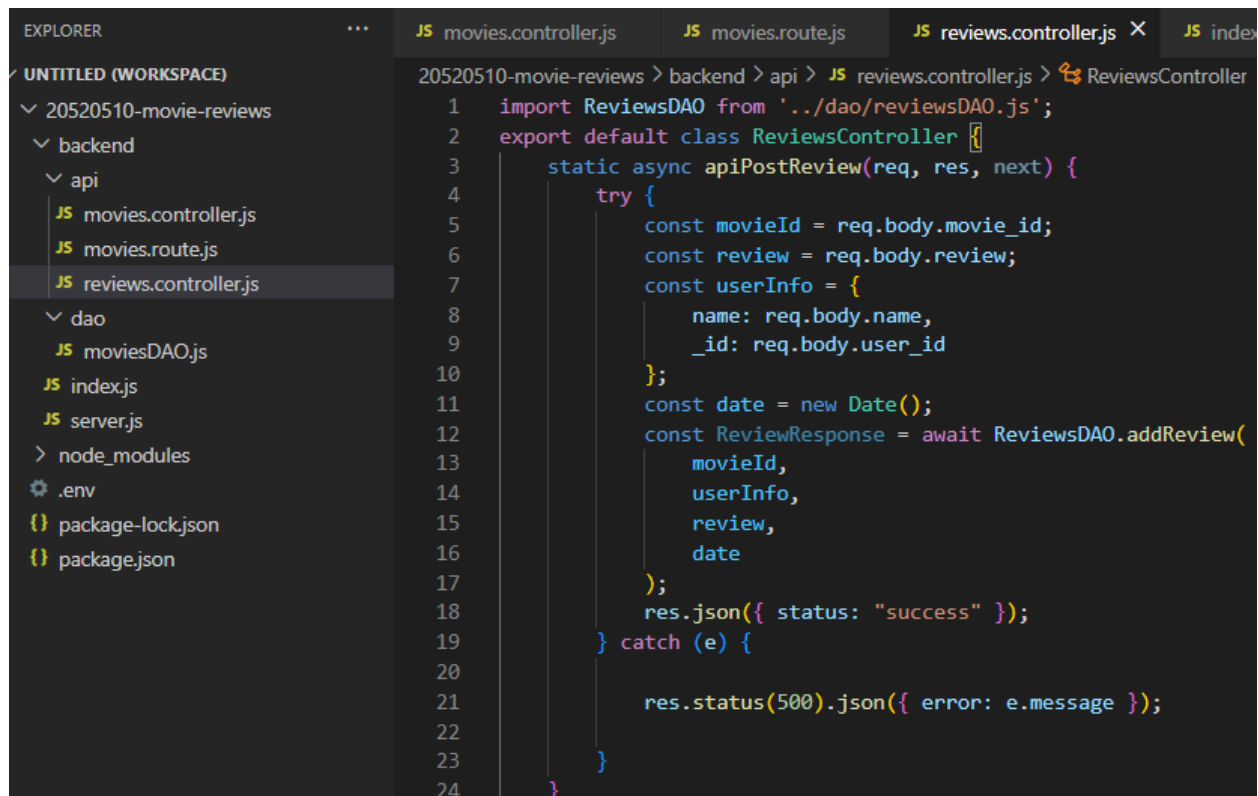


```
EXPLORER
20520510-movie-reviews
  backend
    api
      JS movies.controller.js
      JS movies.route.js
    dao
      JS moviesDAO.js
    JS index.js
    JS server.js
  > node_modules
  .env
  package-lock.json
  package.json

20520510-movie-reviews > backend > api > JS movies.route.js > default
1  import express from 'express';
2  import MoviesController from './movies.controller.js';
3  import ReviewsController from './reviews.controller.js';
4  const router = express.Router(); // get access to express router
5  router.route('/').get(MoviesController.apiGetMovies);
6  router
7    .route('/review')
8      .post(ReviewsController.apiPostReview)
9      .put(ReviewsController.apiUpdateReview)
10     .delete(ReviewsController.apiDeleteReview);
11  export default router;
```

## Bài 2: Thiết lập Controller cho review.

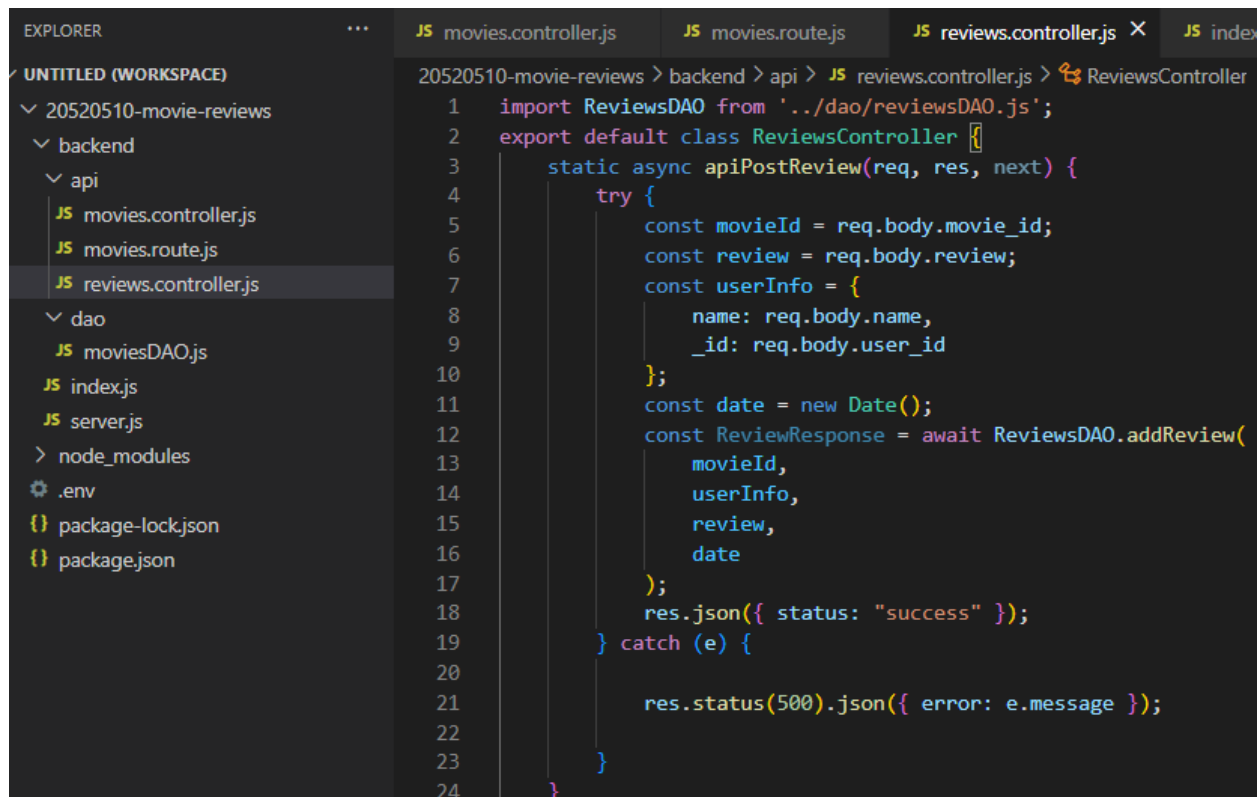
2.1 Tạo tệp tin reviews.controller.js trong thư mục api chứa một class có tên ReviewsController để quản lý các yêu cầu có liên quan đến review từ người dùng gửi lên từ máy khách.



The screenshot shows the VS Code interface. On the left, the Explorer sidebar displays the project structure: '20520510-movie-reviews' > 'backend' > 'api'. The 'api' folder is expanded, showing 'movies.controller.js', 'movies.route.js', and 'reviews.controller.js'. The 'reviews.controller.js' file is selected and its content is displayed in the main editor. The code defines a 'ReviewsController' class with a static method 'apiPostReview'.

```
1 import ReviewsDAO from '../dao/reviewsDAO.js';
2 export default class ReviewsController {
3   static async apiPostReview(req, res, next) {
4     try {
5       const movieId = req.body.movie_id;
6       const review = req.body.review;
7       const userInfo = {
8         name: req.body.name,
9         _id: req.body.user_id
10      };
11       const date = new Date();
12       const ReviewResponse = await ReviewsDAO.addReview(
13         movieId,
14         userInfo,
15         review,
16         date
17       );
18       res.json({ status: "success" });
19     } catch (e) {
20
21       res.status(500).json({ error: e.message });
22     }
23   }
24 }
```

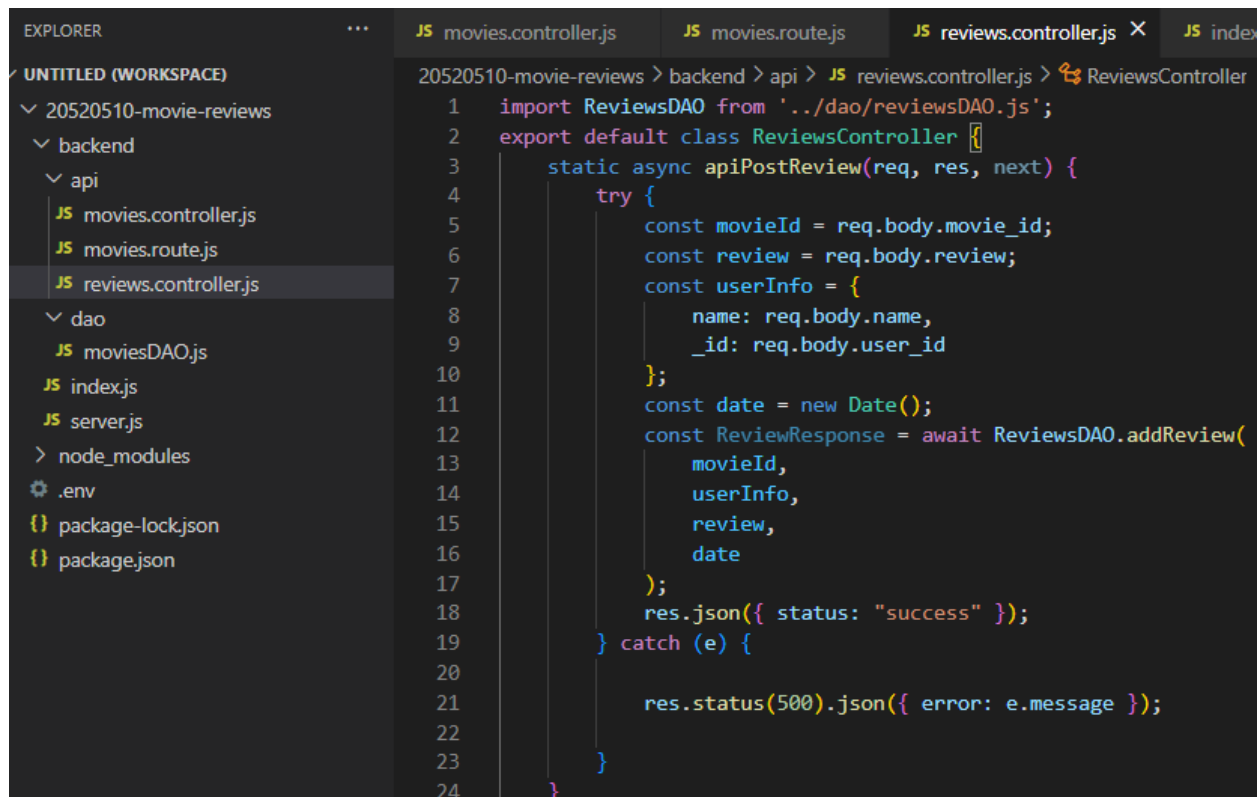
2.2 Trong tệp tin vừa tạo ở bài 2.1 sẽ chứa dòng lệnh import nội dung từ tệp tin reviewsDAO.js (sẽ tạo ở bài 3) để gọi tới các hàm tương tác dữ liệu.



```
1 import ReviewsDAO from '../dao/reviewsDAO.js';
2 export default class ReviewsController {
3   static async apiPostReview(req, res, next) {
4     try {
5       const movieId = req.body.movie_id;
6       const review = req.body.review;
7       const userInfo = {
8         name: req.body.name,
9         _id: req.body.user_id
10      };
11      const date = new Date();
12      const ReviewResponse = await ReviewsDAO.addReview(
13        movieId,
14        userInfo,
15        review,
16        date
17      );
18      res.json({ status: "success" });
19    } catch (e) {
20
21      res.status(500).json({ error: e.message });
22    }
23  }
24 }
```

2.3 Tạo phương thức có tên `apiPostReview()` để quản lý các yêu cầu được gửi từ máy khách (thiết lập ở bài 1.2).

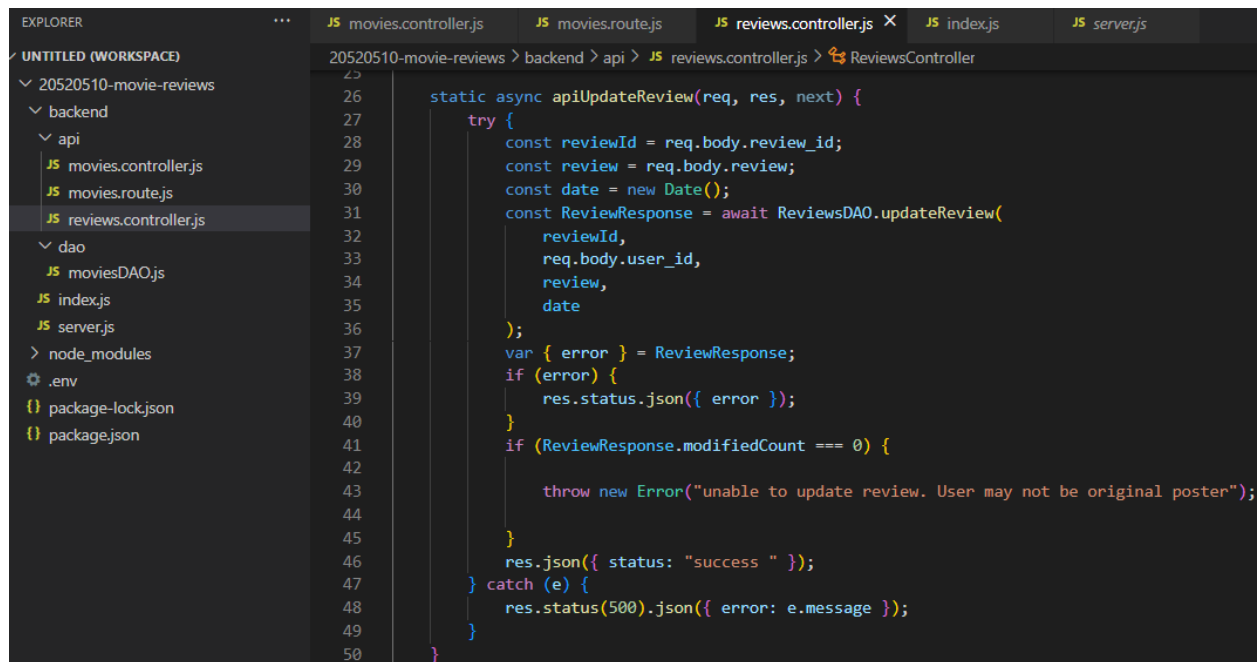
- Phương thức này sẽ lấy dữ liệu gửi lên từ người dùng thông qua tham số `req` gồm có `movie_id`, `review`, `userinfo` gồm `name` và `id` (gửi dưới dạng JSON trong body của `req`), và tạo ra một biến `date` để lưu trữ ngày tháng năm hiện tại của `review`.
- Sau đó, gọi đến hàm `addReview()` được định nghĩa trong `ReviewsDAO` (sẽ tạo ở bài 3) để thêm được `review` vào db.
- Nếu thêm dữ liệu thành công sẽ trả về cho máy khách 1 thông báo dưới dạng JSON báo 'success', nếu không thành công sẽ log lỗi ra trên màn hình console của terminal.



```
1 import ReviewsDAO from '../dao/reviewsDAO.js';
2 export default class ReviewsController {
3   static async apiPostReview(req, res, next) {
4     try {
5       const movieId = req.body.movie_id;
6       const review = req.body.review;
7       const userInfo = {
8         name: req.body.name,
9         _id: req.body.user_id
10      };
11      const date = new Date();
12      const ReviewResponse = await ReviewsDAO.addReview(
13        movieId,
14        userInfo,
15        review,
16        date
17      );
18      res.json({ status: "success" });
19    } catch (e) {
20
21      res.status(500).json({ error: e.message });
22    }
23  }
24 }
```

2.4 Tạo phương thức có tên `apiUpdateReview()` để quản lý các yêu cầu được gửi từ máy khách (thiết lập ở bài 1.3).

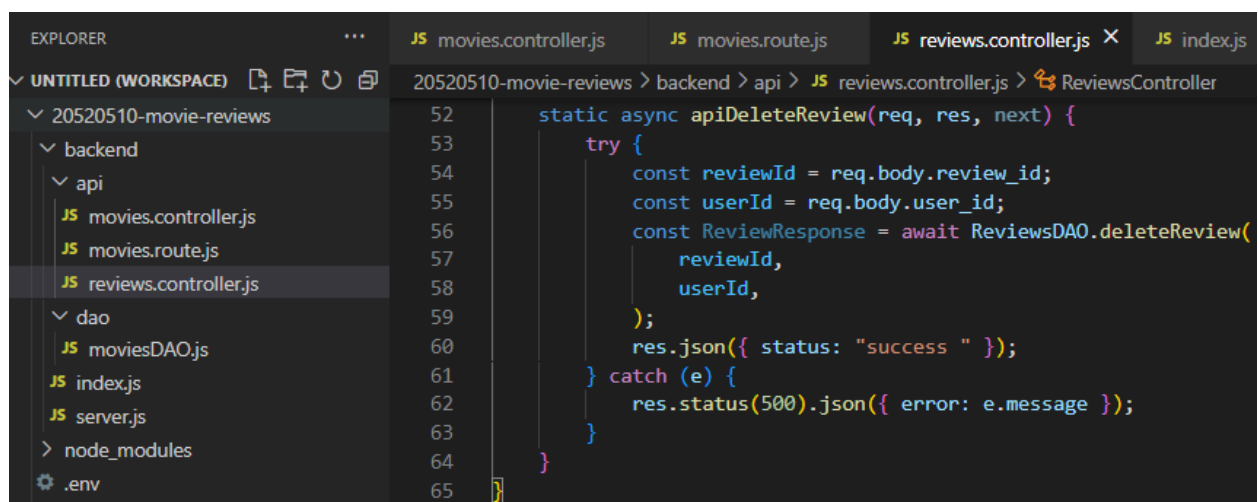
- Phương thức này sẽ lấy dữ liệu gửi lên từ người dùng thông qua tham số `req` gồm có `review_id`, `user_id` (gửi dưới dạng JSON trong body của `req`), và tạo ra một biến `date` để lưu trữ ngày tháng năm hiện tại của review mới, lưu ý, muốn update thành công review thì `id` user phải là user đã tạo ra review.
- Sau đó, gọi đến hàm `updateReview()` được định nghĩa trong `ReviewsDAO` (sẽ tạo ở bài 3) để sửa dữ liệu review trên db.
- Nếu thêm dữ liệu thành công sẽ trả về cho máy khách 1 thông báo dưới dạng JSON báo 'success', nếu không thành công sẽ log lỗi ra trên màn hình console của terminal.
- Lưu ý: Cần tạo ra biến `ReviewResponse` để lấy kết quả trả về khi gọi hàm `updateReview()`, vì hàm `updateReview()` trong bài 3 sẽ có trả về một biến tên là `modifiedCount` để xác định xem là có review nào thực sự đã được chỉnh sửa hay chưa.

A screenshot of a code editor showing the file explorer on the left with the project structure: 20520510-movie-reviews > backend > api > reviews.controller.js. The main editor displays the code for the apiUpdateReview function in reviews.controller.js. The function is static and asynchronous, taking req, res, and next as arguments. It uses a try-catch block to handle the update. Inside the try block, it extracts reviewId and review from req.body, creates a new Date object, and calls ReviewsDAO.updateReview with reviewId, req.body.user\_id, review, and date. It then checks for an error in the ReviewResponse and throws an error if the modifiedCount is 0. Finally, it sends a success response or a 500 error response.

```
26 static async apiUpdateReview(req, res, next) {
27   try {
28     const reviewId = req.body.review_id;
29     const review = req.body.review;
30     const date = new Date();
31     const ReviewResponse = await ReviewsDAO.updateReview(
32       reviewId,
33       req.body.user_id,
34       review,
35       date
36     );
37     var { error } = ReviewResponse;
38     if (error) {
39       res.status(500).json({ error });
40     }
41     if (ReviewResponse.modifiedCount === 0) {
42       throw new Error("unable to update review. User may not be original poster");
43     }
44     res.json({ status: "success " });
45   } catch (e) {
46     res.status(500).json({ error: e.message });
47   }
48 }
49
50 }
```

2.5 Tạo phương thức có tên `apiDeleteReview()` để quản lý các yêu cầu được gửi từ máy khách (thiết lập ở bài 1.3).

- Phương thức này sẽ lấy dữ liệu gửi lên từ người dùng thông qua tham số `req` gồm có `review_id`, `user_id` (gửi dưới dạng JSON trong body của `req`), và thực hiện việc xoá review thông qua hàm `deleteReview` trong lớp `ReviewsDAO` sẽ tạo ở bài 3.

A screenshot of a code editor showing the file explorer on the left with the project structure: 20520510-movie-reviews > backend > api > reviews.controller.js. The main editor displays the code for the apiDeleteReview function in reviews.controller.js. The function is static and asynchronous, taking req, res, and next as arguments. It uses a try-catch block to handle the deletion. Inside the try block, it extracts reviewId and userId from req.body and calls ReviewsDAO.deleteReview with reviewId and userId. It then sends a success response or a 500 error response.

```
52 static async apiDeleteReview(req, res, next) {
53   try {
54     const reviewId = req.body.review_id;
55     const userId = req.body.user_id;
56     const ReviewResponse = await ReviewsDAO.deleteReview(
57       reviewId,
58       userId,
59     );
60     res.json({ status: "success " });
61   } catch (e) {
62     res.status(500).json({ error: e.message });
63   }
64 }
65 }
```

Bài 3: Thiết lập DAO cho reviews.

3.1 Trong thư mục DAO tạo tệp tin `reviewsDAO.js`.

- Tập tin này cần import package mongodb đã cài đặt ở lab 2 để sử dụng một số phương thức cần thiết.
- Tạo một hằng số tên ObjectId = mongodb.ObjectId để sau này xử lý một số tác vụ liên quan đến trường dữ liệu \_id trong mongodb.
- Tạo ra một biến reviews để tham chiếu tới collection reviews sẽ tạo sau trên db.

```

1 import mongodb from "mongodb";
2 const ObjectId = mongodb.ObjectId;
3 let reviews;
4 export default class ReviewsDAO {
5   static async injectDB(conn) {
6     if (reviews) {
7       return;
8     }
9     try {
10      reviews = await conn.db(process.env.MOVIEREVIEWS_NS).collection('reviews');
11    }
12    catch (e) {
13      console.error('unable to establish connection handle in reviewDAO: ${e}');
14    }
15  }
16 }

```

### 3.2 Tạo phương thức có tên injectDB() giúp kết nối tới collection tương ứng trên db.

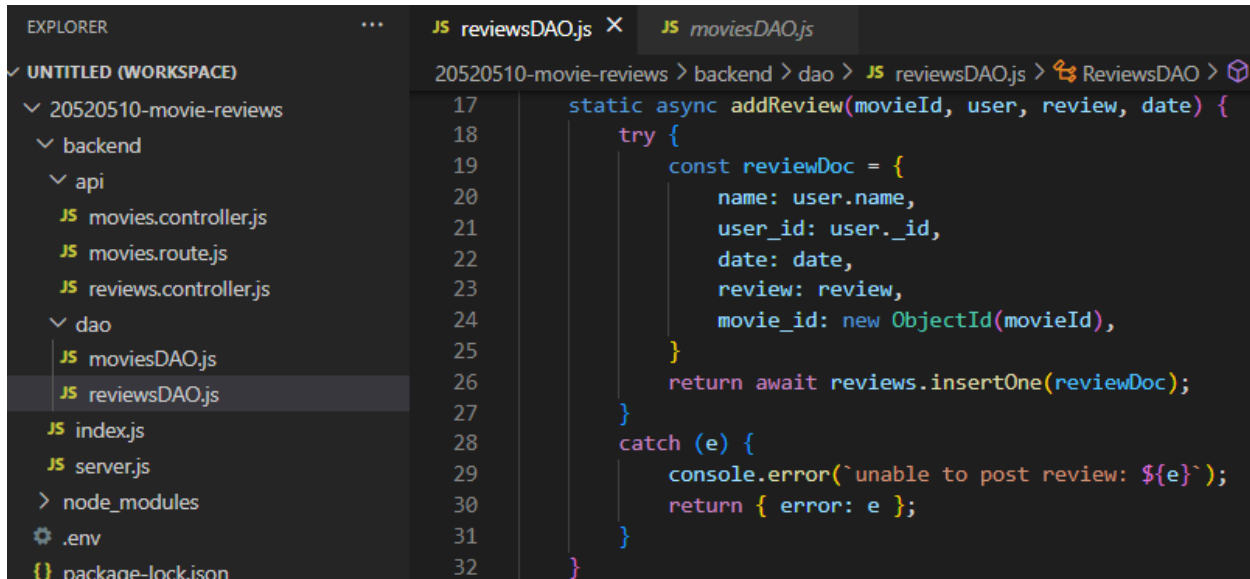
- Cần tạo đối tượng và gọi phương thức injectDB() này trong tập tin index.js để đảm bảo kết nối tới collection thành công, nhằm mục đích tương tác với dữ liệu.
- Lưu ý: việc gọi injectDB() này trong tập tin index phải sau dòng lệnh kết nối tới dữ liệu và trước khi khởi tạo máy chủ web.

```

1 import mongodb from "mongodb";
2 const ObjectId = mongodb.ObjectId;
3 let reviews;
4 export default class ReviewsDAO {
5   static async injectDB(conn) {
6     if (reviews) {
7       return;
8     }
9     try {
10      reviews = await conn.db(process.env.MOVIEREVIEWS_NS).collection('reviews');
11    }
12    catch (e) {
13      console.error('unable to establish connection handle in reviewDAO: ${e}');
14    }
15  }
16 }

```

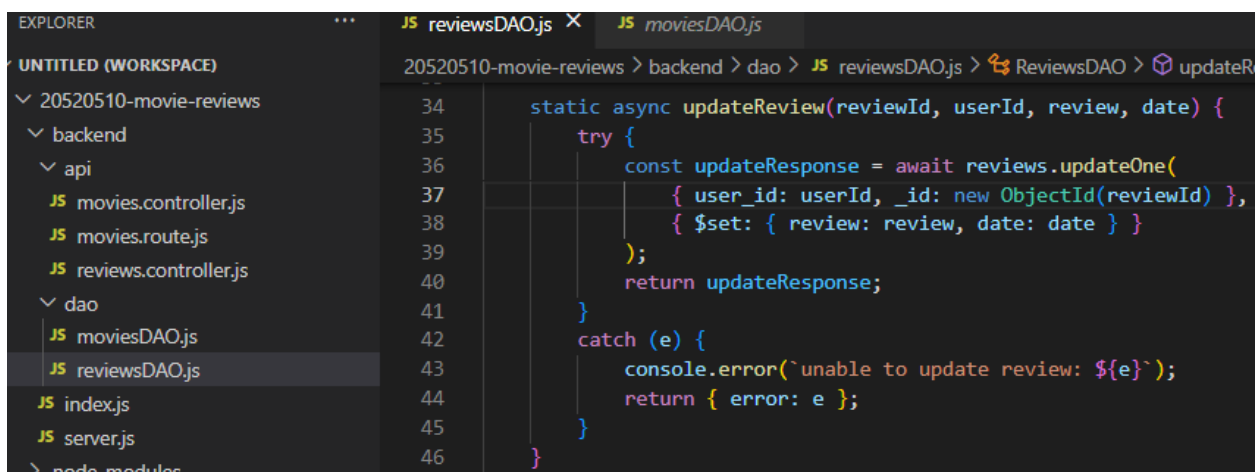
3.3 Tạo phương thức `addReview()` để thêm review vào db, trong hàm này sẽ có gọi một hàm `insertOne()`, lưu ý phải biến chuỗi `movieId` trong tham số truyền vào ở bài 2 thành dạng `ObjectId`.



```
EXPLORER
  20520510-movie-reviews
    backend
      api
        JS movies.controller.js
        JS movies.route.js
        JS reviews.controller.js
      dao
        JS moviesDAO.js
        JS reviewsDAO.js
        JS index.js
        JS server.js
    node_modules
    .env
    package-lock.json

JS reviewsDAO.js
20520510-movie-reviews > backend > dao > JS reviewsDAO.js > ReviewsDAO >
17 static async addReview(movieId, user, review, date) {
18   try {
19     const reviewDoc = {
20       name: user.name,
21       user_id: user._id,
22       date: date,
23       review: review,
24       movie_id: new ObjectId(movieId),
25     }
26     return await reviews.insertOne(reviewDoc);
27   }
28   catch (e) {
29     console.error(`unable to post review: ${e}`);
30     return { error: e };
31   }
32 }
```

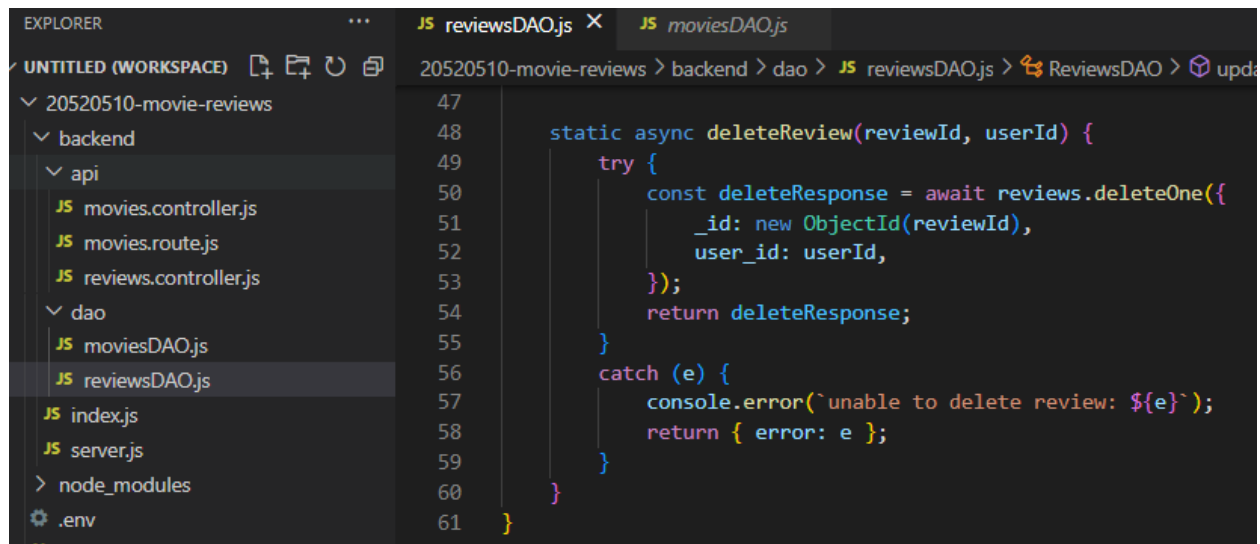
3.4 Tạo phương thức `updateReview()` để sửa review trên db, trong hàm này sẽ có gọi một hàm `updateOne()`, lưu ý phải biến chuỗi `reviewId` trong tham số truyền vào ở bài 2 thành dạng `ObjectId`, phải cùng `userId` mới cho phép sửa review.



```
EXPLORER
  20520510-movie-reviews
    backend
      api
        JS movies.controller.js
        JS movies.route.js
        JS reviews.controller.js
      dao
        JS moviesDAO.js
        JS reviewsDAO.js
        JS index.js
        JS server.js
    node_modules

JS reviewsDAO.js
20520510-movie-reviews > backend > dao > JS reviewsDAO.js > ReviewsDAO > updateR
34 static async updateReview(reviewId, userId, review, date) {
35   try {
36     const updateResponse = await reviews.updateOne(
37       { user_id: userId, _id: new ObjectId(reviewId) },
38       { $set: { review: review, date: date } }
39     );
40     return updateResponse;
41   }
42   catch (e) {
43     console.error(`unable to update review: ${e}`);
44     return { error: e };
45   }
46 }
```

3.5 Tạo phương thức `deleteReview()` để xóa review vào db, trong hàm này sẽ có gọi một hàm `deleteOne()`, lưu ý phải biến chuỗi `reviewId` trong tham số truyền vào ở bài 2 thành dạng `ObjectId`, phải cùng `userId` mới cho phép xóa review.



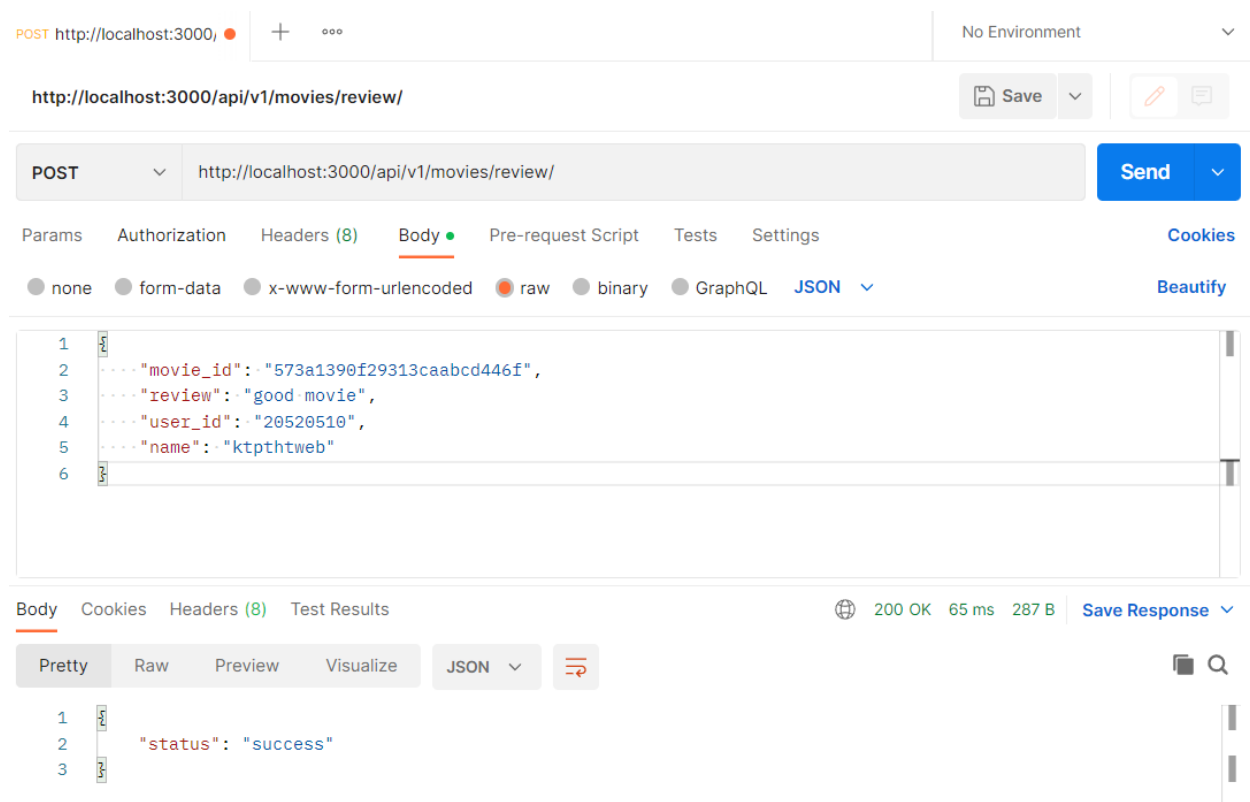
```
47
48 static async deleteReview(reviewId, userId) {
49     try {
50         const deleteResponse = await reviews.deleteOne({
51             _id: new ObjectId(reviewId),
52             user_id: userId,
53         });
54         return deleteResponse;
55     }
56     catch (e) {
57         console.error(`unable to delete review: ${e}`);
58         return { error: e };
59     }
60 }
61 }
```

3.6 Thử nghiệm các API xem đã thành công hay chưa bằng phần mềm hỗ trợ như Insomnia:

Yêu cầu đổi user\_id thành MSSV

Ví dụ: Thêm/Xoá/Sửa dữ liệu (tham khảo slide 60-63 trong bài học)

Thêm dữ liệu:





Kết quả sau khi thêm dữ liệu:

20520510 - IE213

Documents  
sample\_mflix.revi...

sample\_mflix.reviews

1 DOCUMENTS 1 INDEXES

Documents Aggregations Schema Explain Plan Indexes Validation

Filter Type a query: { field: 'value' } Reset Find More Options

ADD DATA EXPORT COLLECTION

1 - 1 of 1

```
{
  "_id": ObjectId("640c4eedccc184f90e18d6f3"),
  "name": "ktpthtweb",
  "user_id": "20520510",
  "date": "2023-03-11T09:50:37.896+00:00",
  "review": "good movie",
  "movie_id": ObjectId("573a1390f29313caabcd446f")
}
```

Sửa dữ liệu:

PUT http://localhost:3000/ No Environment

http://localhost:3000/api/v1/movies/review/ Save

PUT http://localhost:3000/api/v1/movies/review/ Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "review_id": "640c4eedccc184f90e18d6f3",
3   "review": "bad movie",
4   "user_id": "20520510",
5   "name": "ktpthtweb"
6 }
```

Body Cookies Headers (8) Test Results 200 OK 68 ms 288 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "status": "success"
3 }
```

Kết quả sau khi sửa dữ liệu:

20520510 - IE213

Documents  
sample\_mflix.revi...

+

My Queries

Databases

Search

admin

config

local

sample\_airbnb

sample\_analytics

sample\_geospatial

sample\_guides

sample\_mflix

comments

movies

reviews

sample\_mflix.reviews

1 DOCUMENTS 1 INDEXES

Documents

Aggregations

Schema

Explain Plan

Indexes

Validation

Filter

Type a query: { field: 'value' }

Reset

Find

More Options

ADD DATA

EXPORT COLLECTION

1 - 1 of 1

```

_id: ObjectId('640c4eedccc184f90e18d6f3')
name: "ktpthtweb"
user_id: "20520510"
date: 2023-03-11T09:57:57.736+08:00
review: "bad movie"
movie_id: ObjectId('573a1390f29313caabcd446f')

```

Xóa dữ liệu:

DEL http://localhost:3000/

+

...

No Environment

http://localhost:3000/api/v1/movies/review/

Save

DELETE

http://localhost:3000/api/v1/movies/review/

Send

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

Beautify

```

1  {
2    "review_id": "640c4eedccc184f90e18d6f3",
3    "user_id": "20520510"
4  }

```

Body

Cookies

Headers (8)

Test Results

200 OK 55 ms 288 B

Save Response

Pretty

Raw

Preview

Visualize

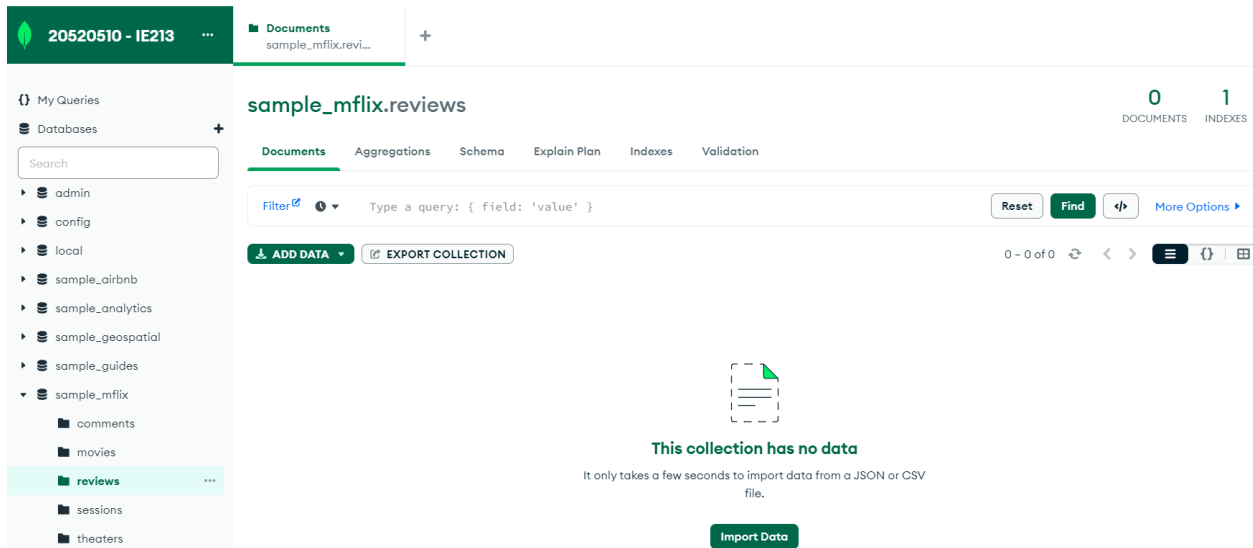
JSON

```

1  {
2    "status": "success "
3  }

```

Kết quả sau khi xóa dữ liệu:



Bài 4: Hoàn thành back-end cho ứng dụng mình họa.

4.1 Thêm 2 định tuyến cho người dùng sử dụng các chức năng sau:

1. Lấy tất cả thông tin của phim và các review có liên quan dựa trên Id của phim.
2. Lấy tất cả các loại rating của phim trên dữ liệu.

```
EXPLORER
20520510-movie-reviews
  backend
    api
      JS movies.controller.js
      JS movies.route.js
      JS reviews.controller.js
    dao
      JS moviesDAO.js
      JS reviewsDAO.js
      JS index.js
      JS server.js

JS movies.route.js
20520510-movie-reviews > backend > api > JS movies.route.js > ...
1 import express from 'express';
2 import MoviesController from './movies.controller.js';
3 import ReviewsController from './reviews.controller.js';
4 const router = express.Router(); // get access to express router
5 router.route('/').get(MoviesController.apiGetMovies);
6 router.route("/id/:id").get(MoviesController.apiGetMovieById);
7 router.route("/ratings").get(MoviesController.apiGetRatings);
8 router
9   .route('/review')
10    .post(ReviewsController.apiPostReview)
11    .put(ReviewsController.apiUpdateReview)
12    .delete(ReviewsController.apiDeleteReview);
13 export default router;
```

4.2 Thêm 2 phương thức controller tương ứng cho phần 4.1 là apiGetMovieById() và apiGetRatings() trong movie controller.

```
31 static async apiGetMovieById(req, res, next) {
32   try {
33     let id = req.params.id || {};
34     let movie = await MoviesDAO.getMovieById(id);
35     if (!movie) {
36       res.status(404).json({ error: "not found" });
37       return;
38     }
39     res.json(movie);
40   }
41   catch (e) {
42     console.log(`api, ${e}`);
43     res.status(500).json({ error: e });
44   }
45 }
46
47 static async apiGetRatings(req, res, next) {
48   try {
49     let propertyTypes = await MoviesDAO.getRatings();
50     res.json(propertyTypes);
51   }
52   catch (e) {
53     console.log(`api, ${e}`);
54     res.status(500).json({ error: e });
55   }
56 }
```

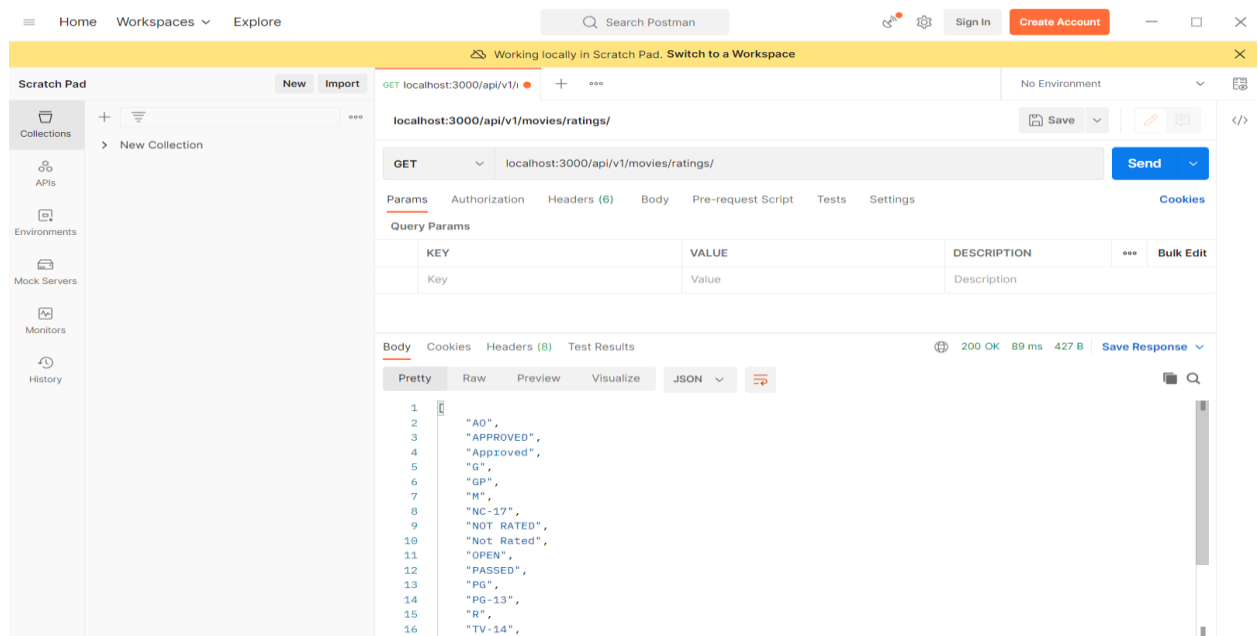
4.3 Thêm 2 phương thức DAO tương ứng cho phần 4.2 là `getRatings()` và `getMovieById()` trong dao movie.

Lưu ý: phần `getMovieById()` sẽ sử dụng một số toán tử và phương thức đã học ở bài thực hành 1 MongoDB như `$match`, `$lookup` (giống khóa ngoại trong SQL) và `aggregate()` để tổng hợp dữ liệu từ nhiều collection.

```
41 }
42 static async getRatings() {
43   let ratings = []
44   try {
45     ratings = await movies.distinct("rated");
46     return ratings;
47   }
48   catch (e) {
49     console.error(`unable to get ratings, ${e}`);
50     return ratings;
51   }
52 }
53 static async getMovieById(id) {
54   try {
55     return await movies.aggregate([
56       {
57         $match: { _id: new ObjectId(id), }
58       },
59       {
60         $lookup: {
61           from: 'reviews', localField: '_id',
62           foreignField: 'movie_id', as: 'reviews',
63         }
64       }
65     ]).next();
66   }
67   catch (e) {
68     console.error(`something went wrong in getMovieById: ${e}`);
69     throw e;
70   }
71 }
```

## 4.4 Thử nghiệm các API vừa tạo ở trên.

Thử nghiệm apiGetRatings() :



Thử nghiệm getMovieById():

GET localhost:3000/api/v1/

+ ...

No Environment

localhost:3000/api/v1/movies/id/573a1390f29313caabcd6223/

Save

GET

localhost:3000/api/v1/movies/id/573a1390f29313caabcd6223/

Send

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body

Cookies

Headers (8)

Test Results

200 OK 64 ms 1.82 KB

Save Response

Pretty

Raw

Preview

Visualize

JSON

```
52         "lastUpdated": "2019-08-21T10:00:29.000Z",
53     },
54     "num_mflix_comments": 0,
55     "reviews": [
56         {
57             "_id": "640c5583f65a6686acac2ab0",
58             "name": "ie213abc",
59             "user_id": "20520510",
60             "date": "2023-03-11T10:18:43.976Z",
61             "review": "nice!",
62             "movie_id": "573a1390f29313caabcd6223"
63         },
```