**Instructions**

**Please read the instructions carefully before answering the questions:**

* You can use the materials on your computer, notebooks, and textbooks.
* You are **NOT allowed** to use electronic devices and other methods to share data with others.
* Do not arbitrarily disconnect the **FPTU\_EXAMONLINE** network while doing the test. If any network problems occur, immediately notify the exam supervisor for assistance.

**In addition to the above conditions, students must fulfill the following requirements:**

1. Use **Visual Studio Code** (VS Code), **MongoDB Compass, Postman** tools to do the exam
2. Use the available browser software on your computer

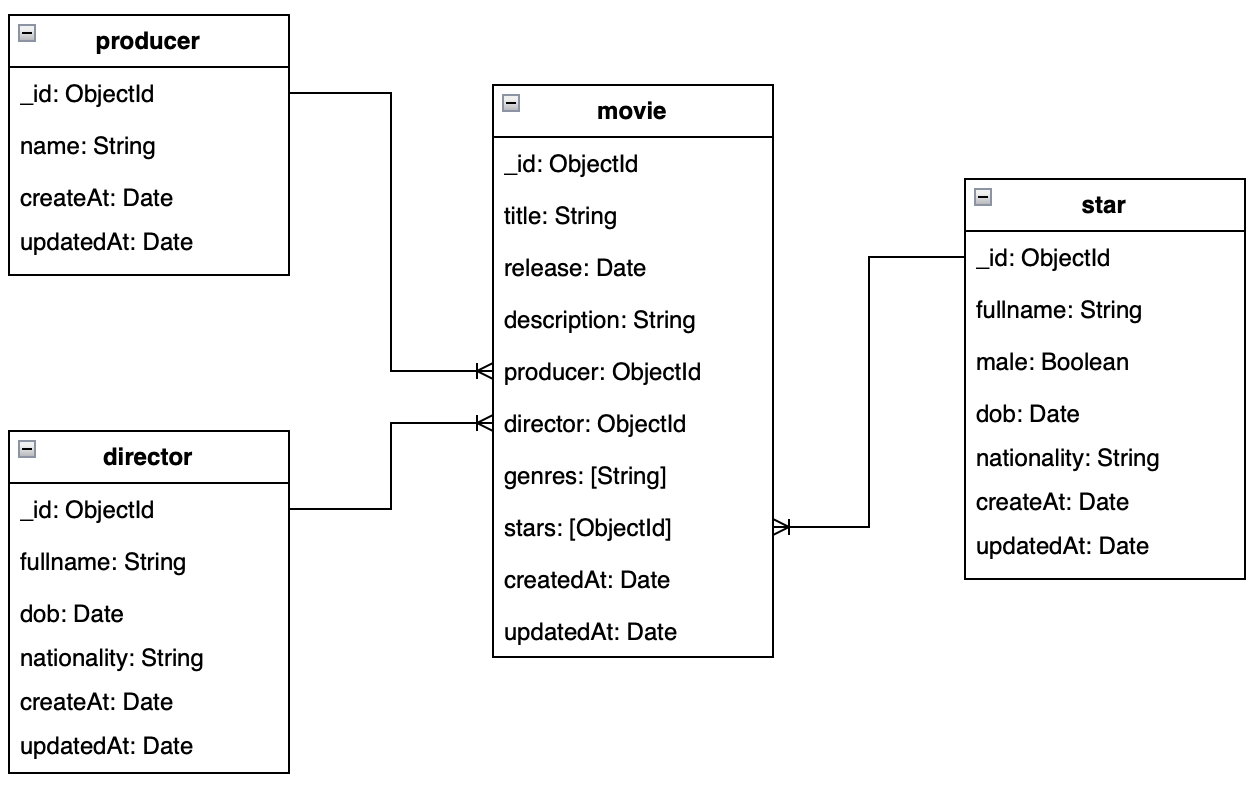
**This part is very important, please read carefully and follow the instructions**

* You are provided with the files containing data in the **given** directory.
* You must use the solution provided in advance via email and install the additional library included in the accompanying request.
* You are **not allowed** to download any additional libraries while taking the test
* Violating one of the above, your exam is considered invalid

**Instructions for submission**

* Before submitting the solution, delete the [**node\_modules**] folders in your test to reduce the space, to meet the allowed capacity of the **PEA\_Client** software
* Use **PEA\_Client** software to browse the **resource** directory to complete the submission process.

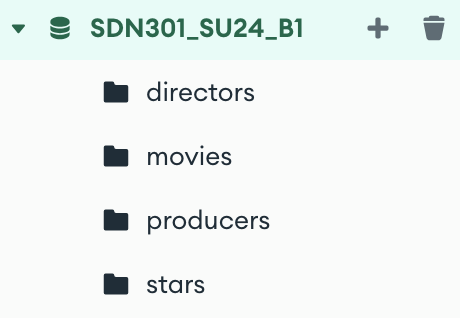
**Use the following database schema for doing this exam:**



**­­­­**

**Requirements:**

Using **MongoDB Compass**, create a database named “**SDN301\_SU24\_B1**”. Create collections corresponding to the data in the **given** directory. Then, import data into the collections created in the database.



Using the given, build a RESTFul API application that performs the requirements of the problem:

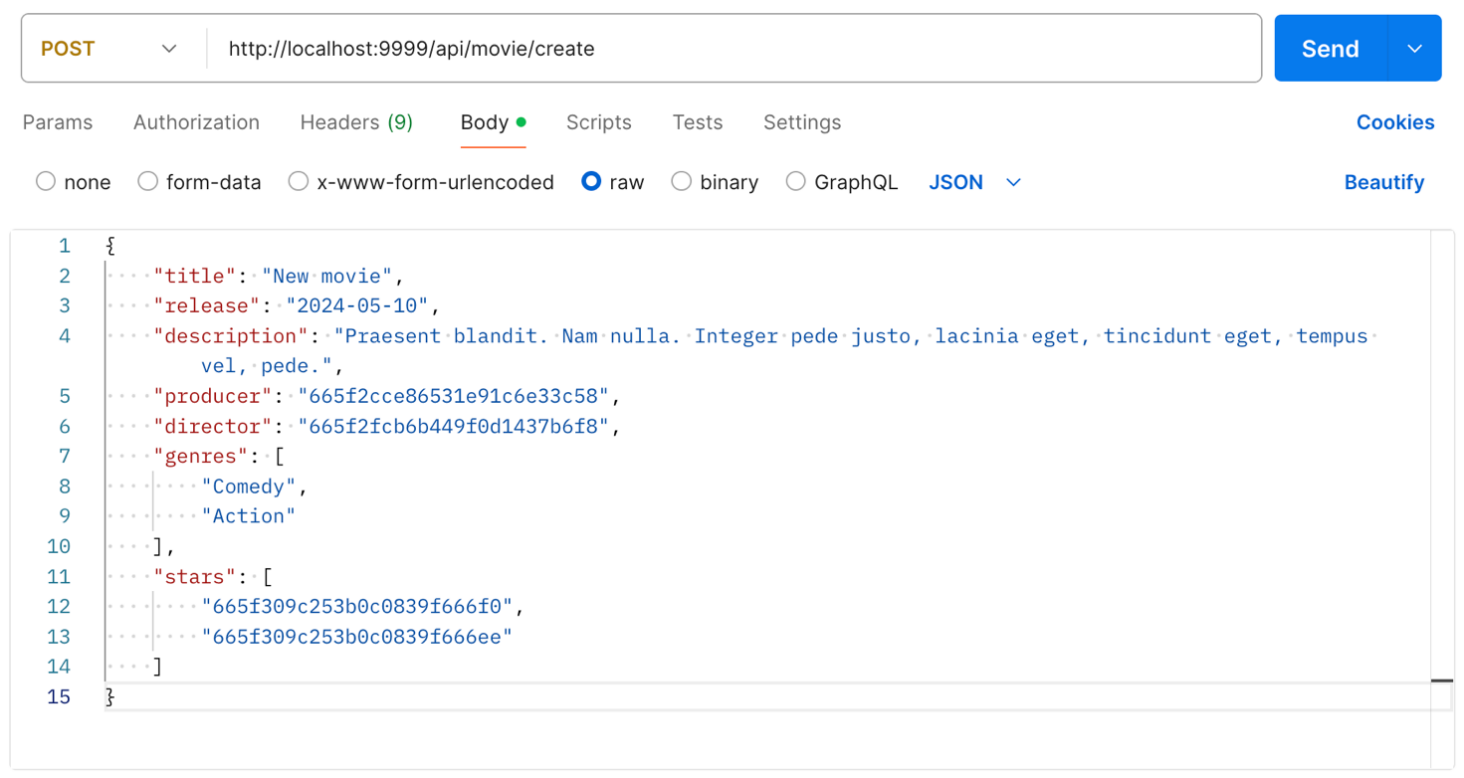
**Note: 0 will be given to the work that**

* Not using connection string to MongoDB in the **.env** file
* Not configuring the root path of RESTFul API Web App at: **http://localhost: 9999**

**Question 1.**

Create a new movie as shown in *Figure 1*, with the following API end-point parameters:

* URL: <http://localhost:9999/api/movie/create>
* Method: POST
* Input: a new movie object
* Example:



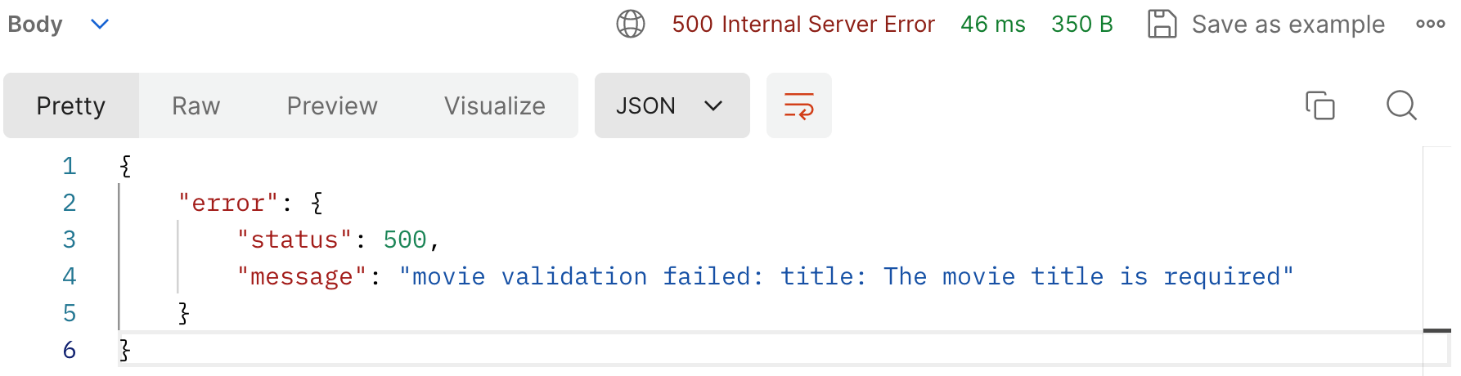
*Figure 1 - Send a request to create a new movie*

* The creation process was successful, the movie was saved to the database and returned data as shown in *Figure 2* - (**1 mark**).



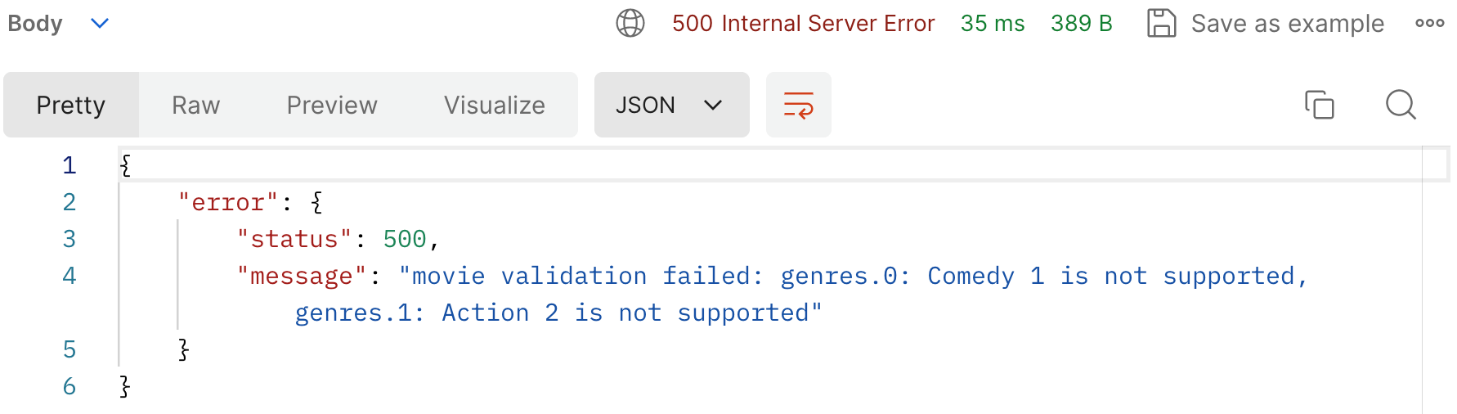
*Figure 2 - Receive results response from the API after creating success*

* When creating a new movie without a **title** field, the result is shown in *Figure 3* - (**0.25 marks**).



*Figure 3 - Returns an error object when the title field is missing*

* The movie genres only accept values ​​including: "**Action**, **Drama**, **Comedy**, **Cartoon**". If the data is incorrect, an error will be displayed as shown in *Figure 4* - (**0.25 marks**).

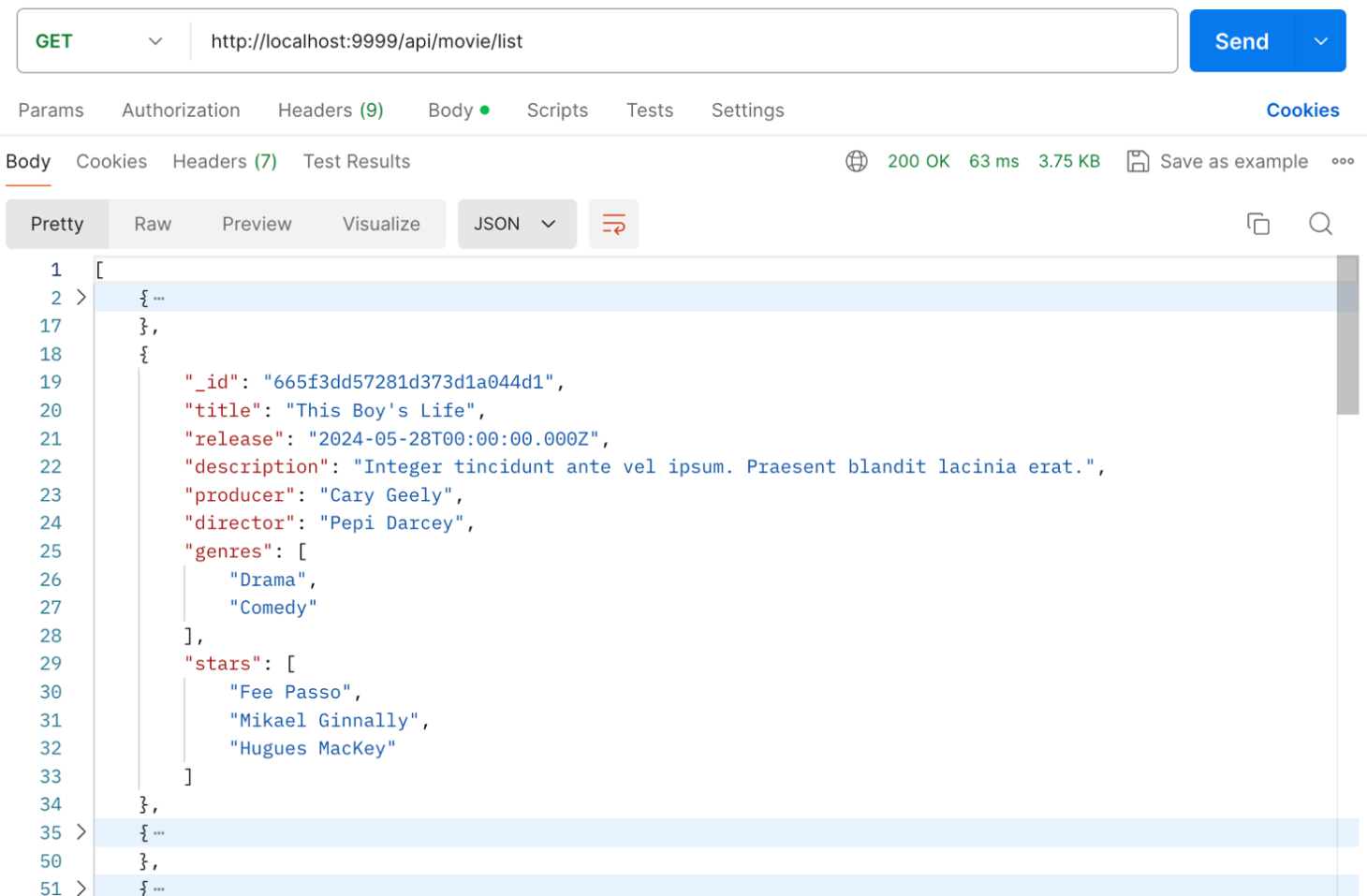


*Figure 4 - Returns an error object when the genres field has an incorrect value*

**Question 2**.

List all movies as shown in *Figure 5*, with the following API endpoint parameters:

* URL: <http://localhost:9999/api/movie/list>
* Method: GET
* Example:



*Figure 5 - List of movies returned successfully*

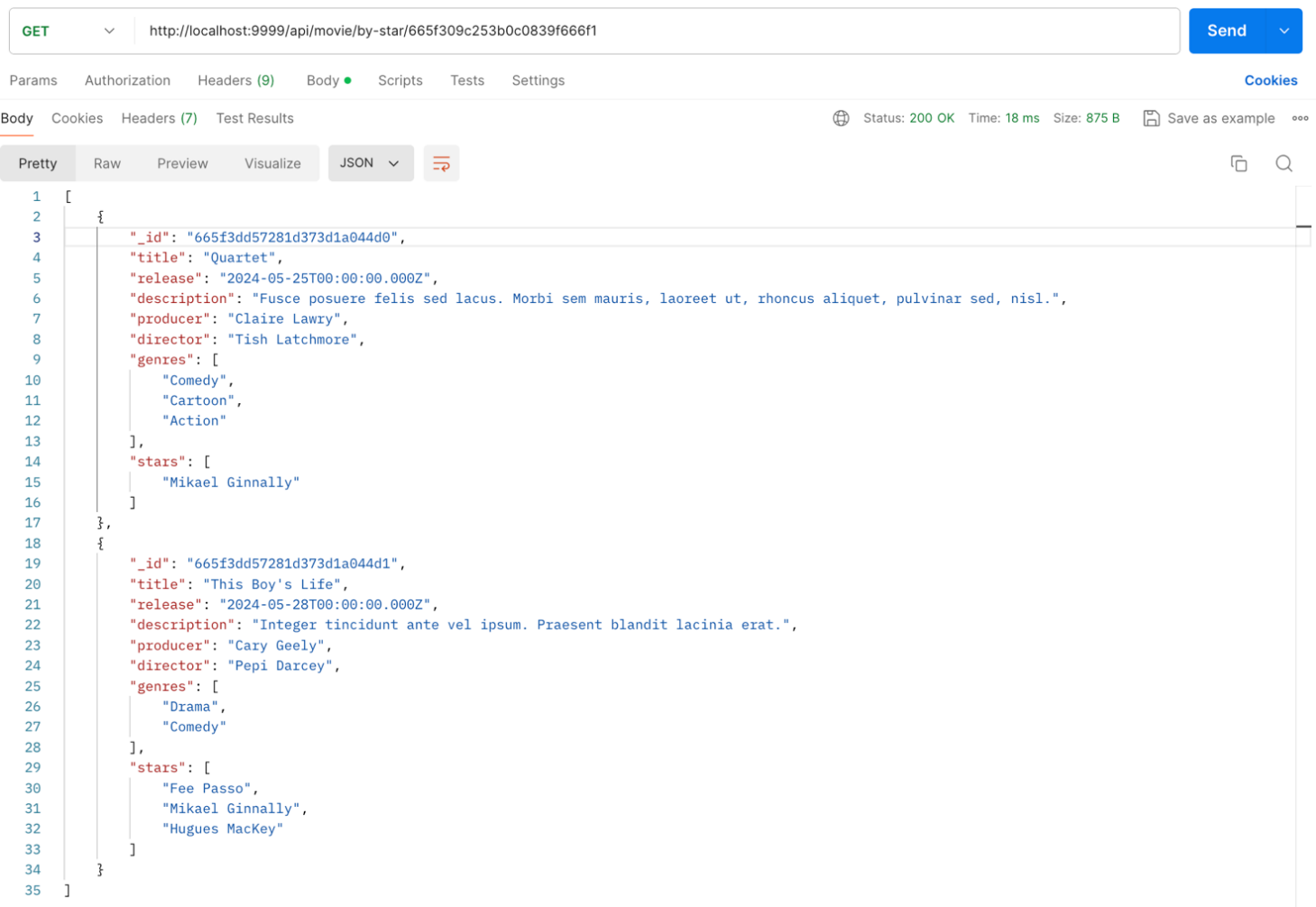
* Returns an array containing all the movies from the database - (**1 mark**).
* Display correct information of fields: **director** and **producer** - (**0.5 marks**).
* Display correct information of the **stars** field - (**0.5 marks**).

­­­­­

­­­­**Question 3**.

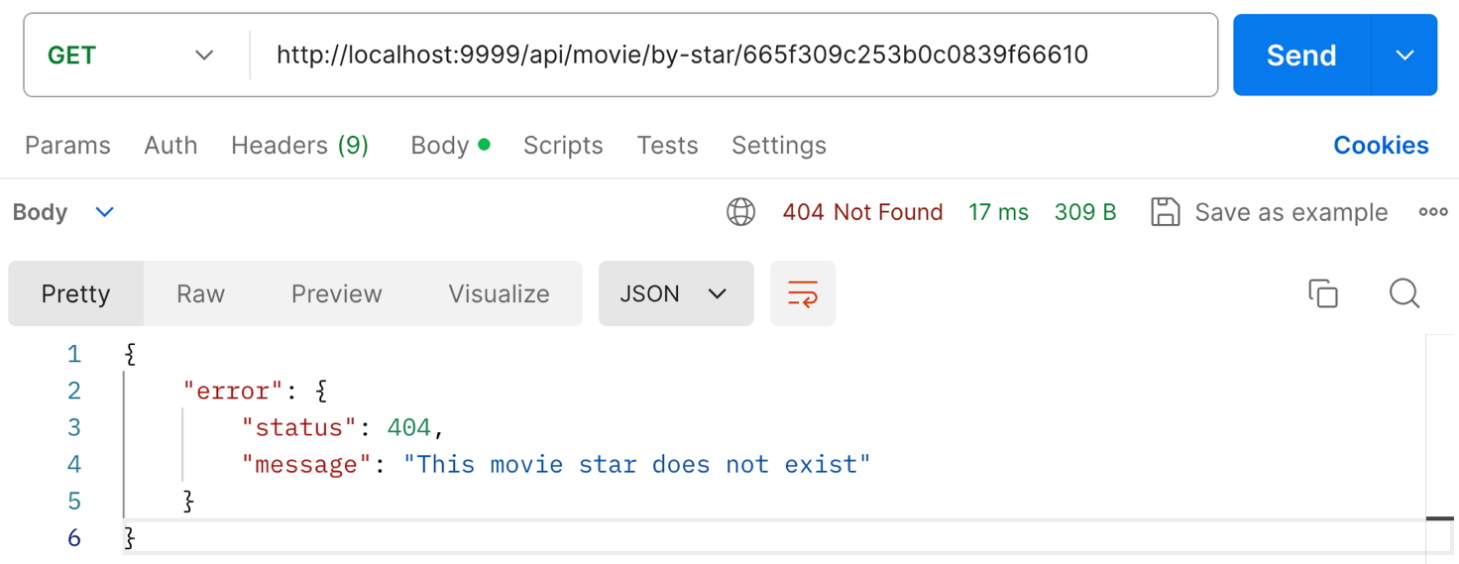
List all the movies that the star has played in as shown in *Figure 6*, with the following API endpoint parameters:

* URL: [http://localhost:9999/api/movie/by-star/**:starId**](http://localhost:9999/api/movie/by-star/:starId)
* Method: GET
* Example:



*Figure 6 - Movies that the star with \_id = “****665f309c253b0c0839f666f1****” has participated in*

* Return all movies as required by the question - (**1 mark**).
* Display correct information of fields: **director** and **producer** - (**0.5 marks**).
* Display correct information of the **stars** field - (**0.5 marks**).
* If the star’s **\_id** is incorrect, return an error as shown in *Figure 7* - (**0.5 marks**).



*Figure 7 - The star’s \_id is incorrect*

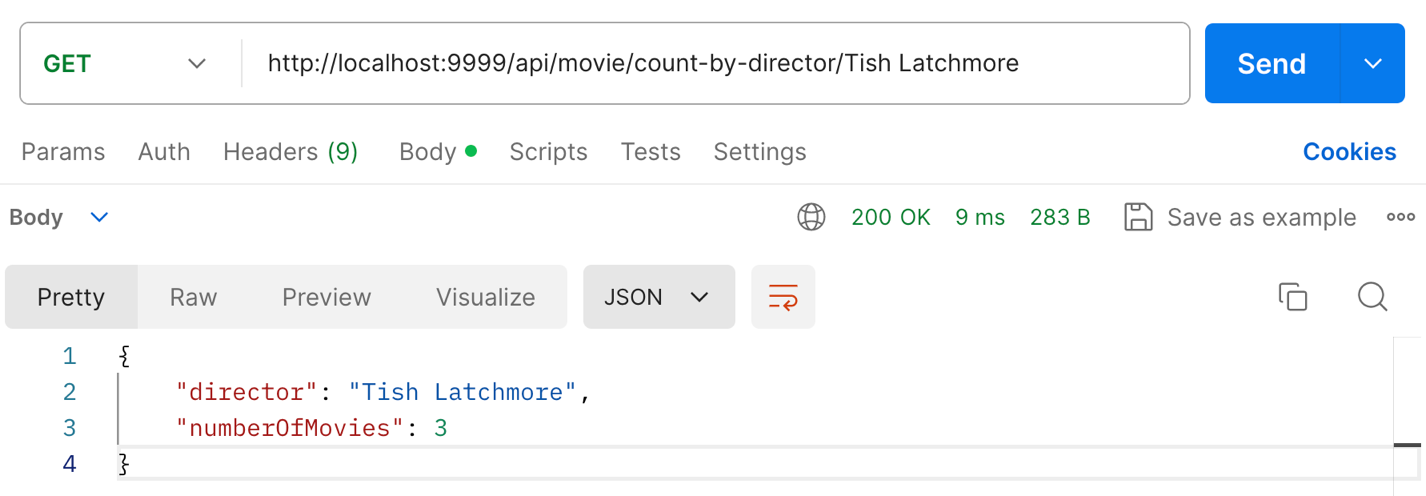
­­­­

**Question 4**.

Statistics of the number of movies by director name as shown in *Figure 8*, with the following API endpoint parameters:

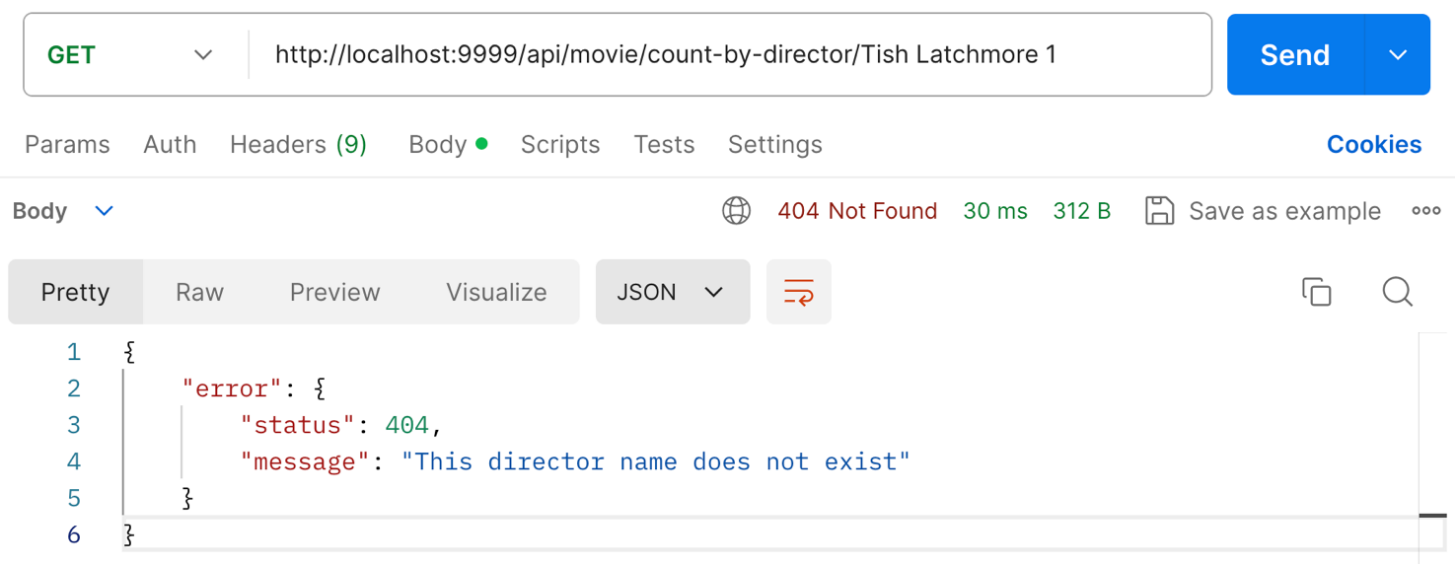
* URL: [http://localhost:9999/api/movie/count-by-director/**:directorName**](http://localhost:9999/api/movie/count-by-director/:directorName)
* Method: GET
* Example:

­­­



*Figure 8 - Number of films produced by director “****Tish Latchmore****”*

* Correct number of movies by director name - (**1 mark**).
* Display the correct format of the returned object - (**0.5 marks**).
* If the director name does not exist, return the result as shown in *Figure 9* - (**0.5 marks**).

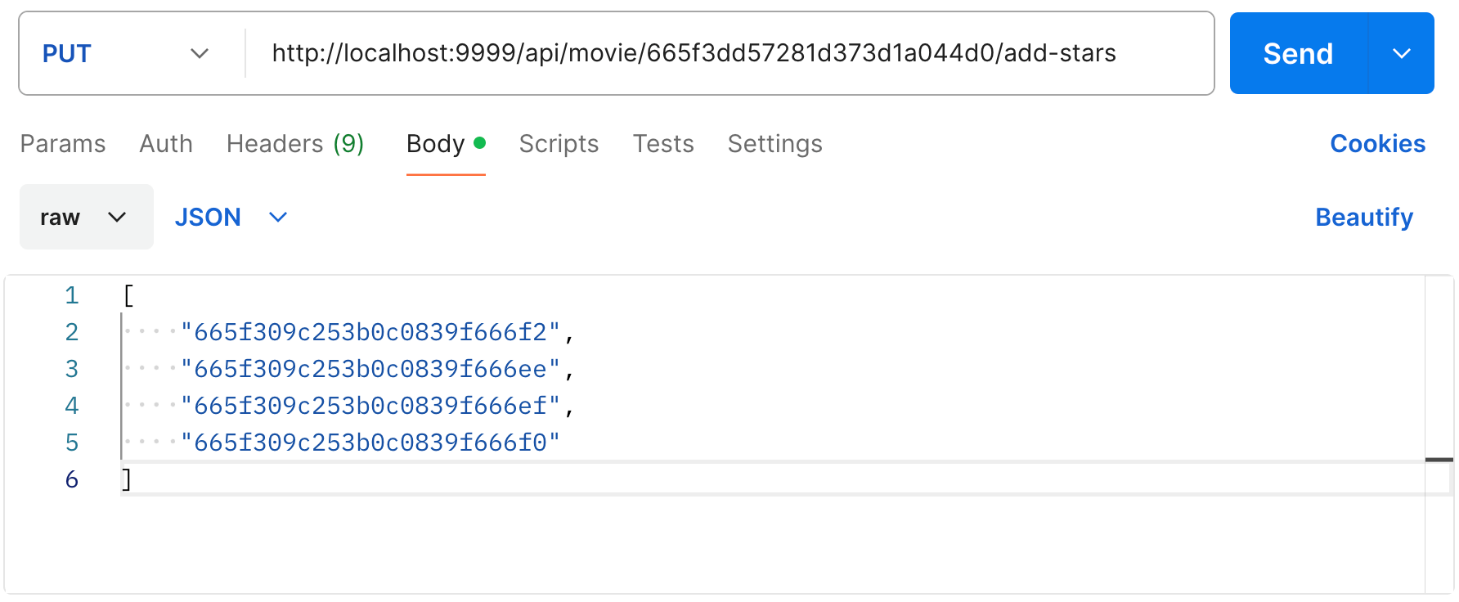


*Figure 9 - The director name does not exist*

**Question 5**.

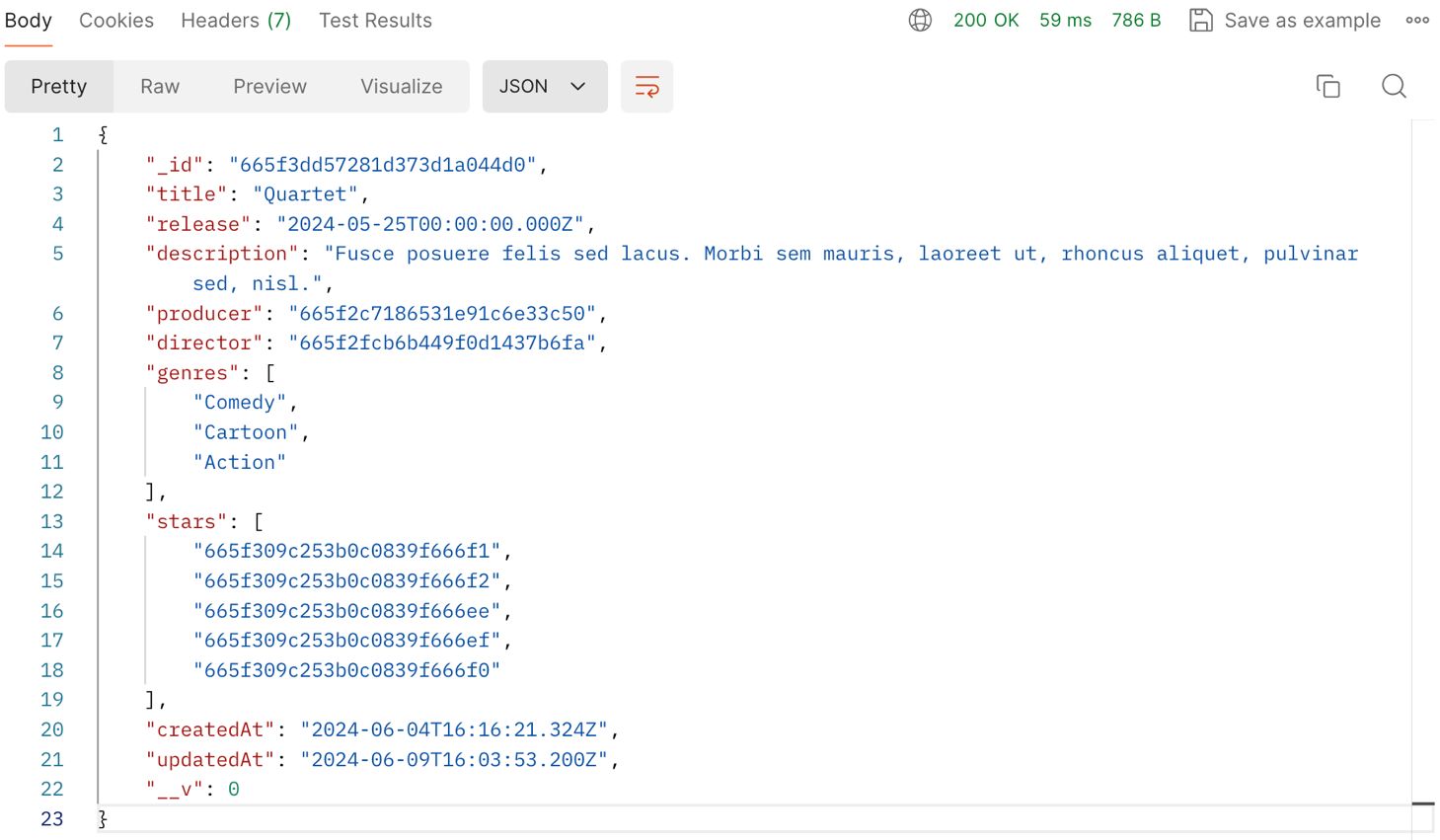
Add the stars to the corresponding movie according to **\_id** as shown in *Figure 10*. If the stars already exist in this movie then only add new stars. With the following API endpoint parameters:

* URL: [http://localhost:9999/api/movie/**:movieId**/add-stars](http://localhost:9999/api/movie/:movieId/add-stars)
* Method: PUT
* Input: an array of star’s \_id
* Example:



*Figure 10 - Add the stars to movie with \_id = “665f3dd57281d373d1a044d0”*

* The process of adding movie stars is successful, returning the movie information including **stars** after saving to the database as shown in *Figure 11* - (**2 marks**).



*Figure 11 - The movie after add the stars*

**--- END ---**