# myFlix web app

## Backend development process

`auth.js`

`index.js`

`models.js`

`passport.js`

`package-lock.json`

`package.json`

| June | 2023 |
|---|---|

*All usernames, passwords or other similar information shown in this presentation is fictive and for illustrative purpose only

## **TABLE** OF CONTENTS

# **TABLE** OF CONTENTS

## 01

## **Setting up** the Node.js development environment

Skills used

Research

## **WHY** WAS THIS STEP IMPORTANT

Organizing the development / coding environment for efficient and well-oriented work from the beginning is not only necessary, but is also one of the keys to ensure smooth workflow and limit avoidable time loss due to inefficient project initialization.

## **WHAT** WAS THE GOAL

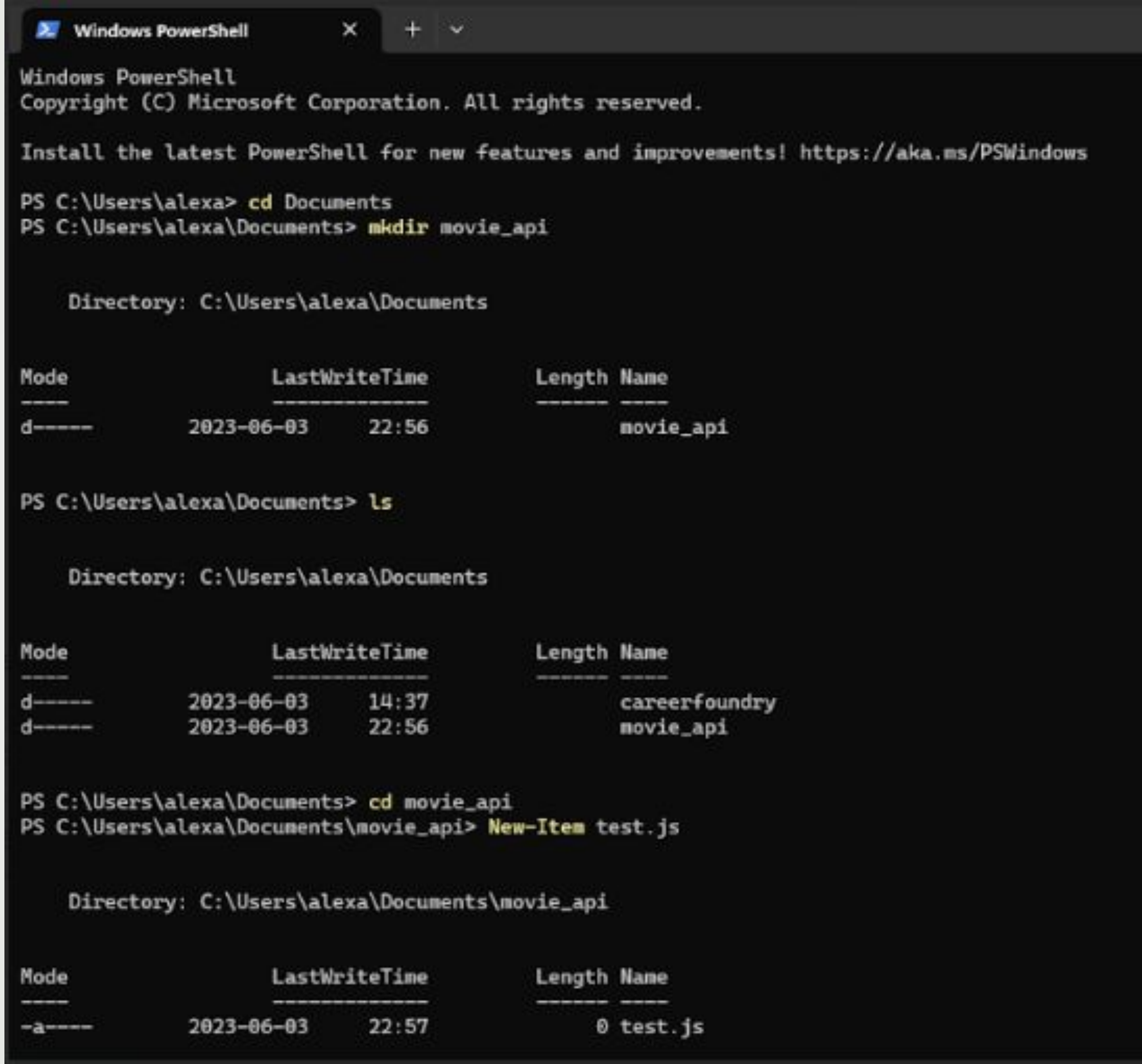
Creating the project directory.

Getting used to working with the Command Line Interface (CLI) / terminal of Windows PowerShell. Installing Node.js via the version management tool nvm (Node Version Manager).

Getting used to working with Node.js own CLI shell program Repl (Node Console).

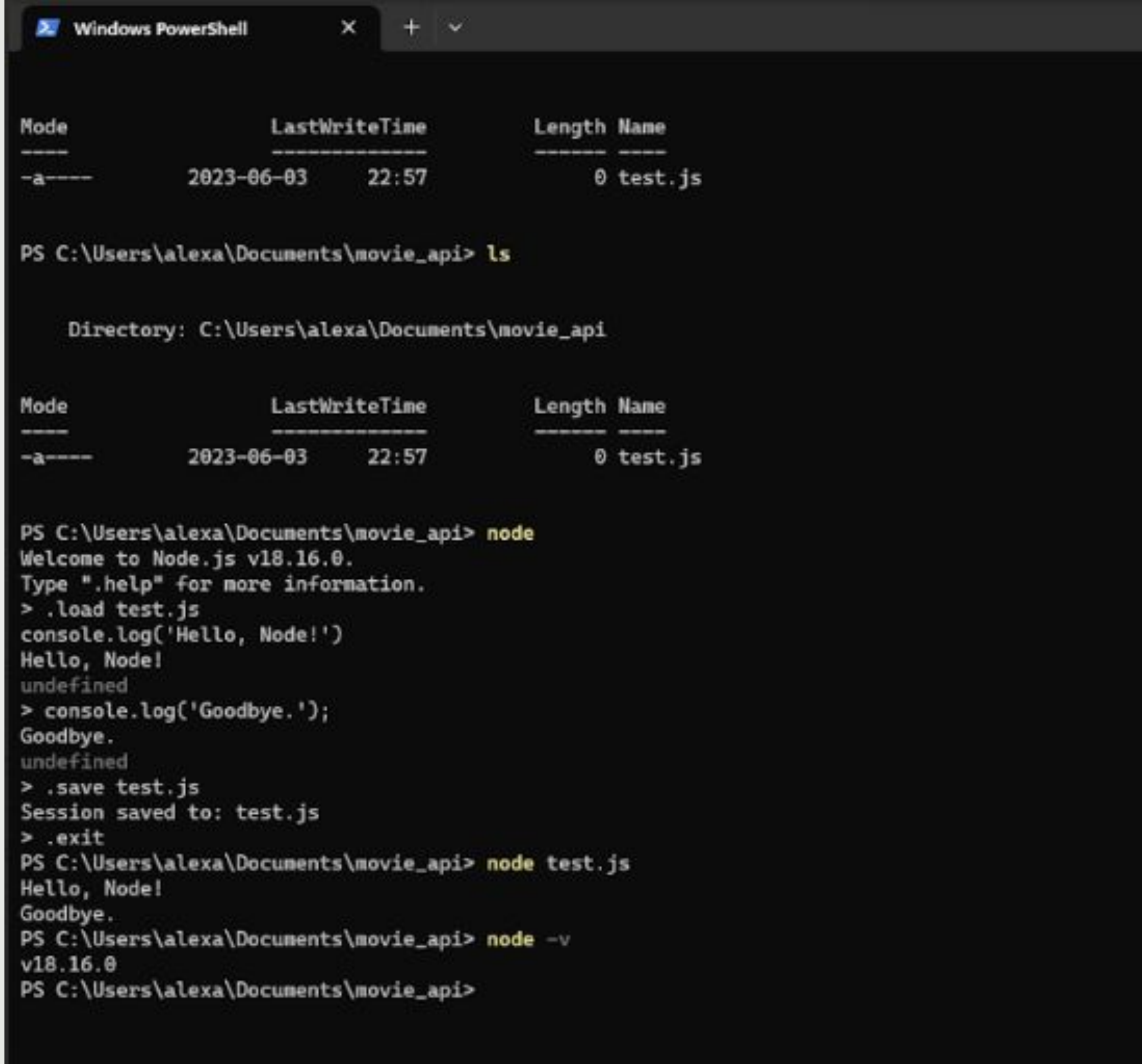
Creating a Github repository for the project.

## **CREATION** OF PROJECT DIRECTORY



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\alexa> cd Documents
PS C:\Users\alexa\Documents> mkdir movie_api


    Directory: C:\Users\alexa\Documents


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        2023-06-03     22:56                movie_api


PS C:\Users\alexa\Documents> ls


    Directory: C:\Users\alexa\Documents


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        2023-06-03     14:37                careerfoundry
d-----        2023-06-03     22:56                movie_api


PS C:\Users\alexa\Documents> cd movie_api
PS C:\Users\alexa\Documents\movie_api> New-Item test.js


    Directory: C:\Users\alexa\Documents\movie_api


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        2023-06-03     22:57              0 test.js
```

## **TEST AND VALIDATION** THAT NODE.JS IS WELL INSTALLED



```
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        2023-06-03     22:57              0 test.js


PS C:\Users\alexa\Documents\movie_api> ls


    Directory: C:\Users\alexa\Documents\movie_api


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        2023-06-03     22:57              0 test.js


PS C:\Users\alexa\Documents\movie_api> node
Welcome to Node.js v18.16.0.
Type ".help" for more information.
> .load test.js
console.log('Hello, Node!')
Hello, Node!
undefined
> console.log('Goodbye.');
Goodbye.
undefined
> .save test.js
Session saved to: test.js
> .exit
PS C:\Users\alexa\Documents\movie_api> node test.js
Hello, Node!
Goodbye.
PS C:\Users\alexa\Documents\movie_api> node -v
v18.16.0
PS C:\Users\alexa\Documents\movie_api>
```

02

HTTP

URL

FS

## **Using** built-in Node modules to return files to users and log requests

Skills used

Research
Problem solving
Code writing

## **WHY** WAS THIS STEP IMPORTANT

Learning about Node different modules types (built-in, user-defined and third party) was important to understand the very structure of Node.js and how it works. Then, learning more specifically about Node built-in modules was necessary to (1) get to know how to use some of them (HTTP, URL, FS) and (2) understand how multiple modules can be used / work together.

```
const http = require('http'),
 url = require('url'),
 fs = require('fs');
```

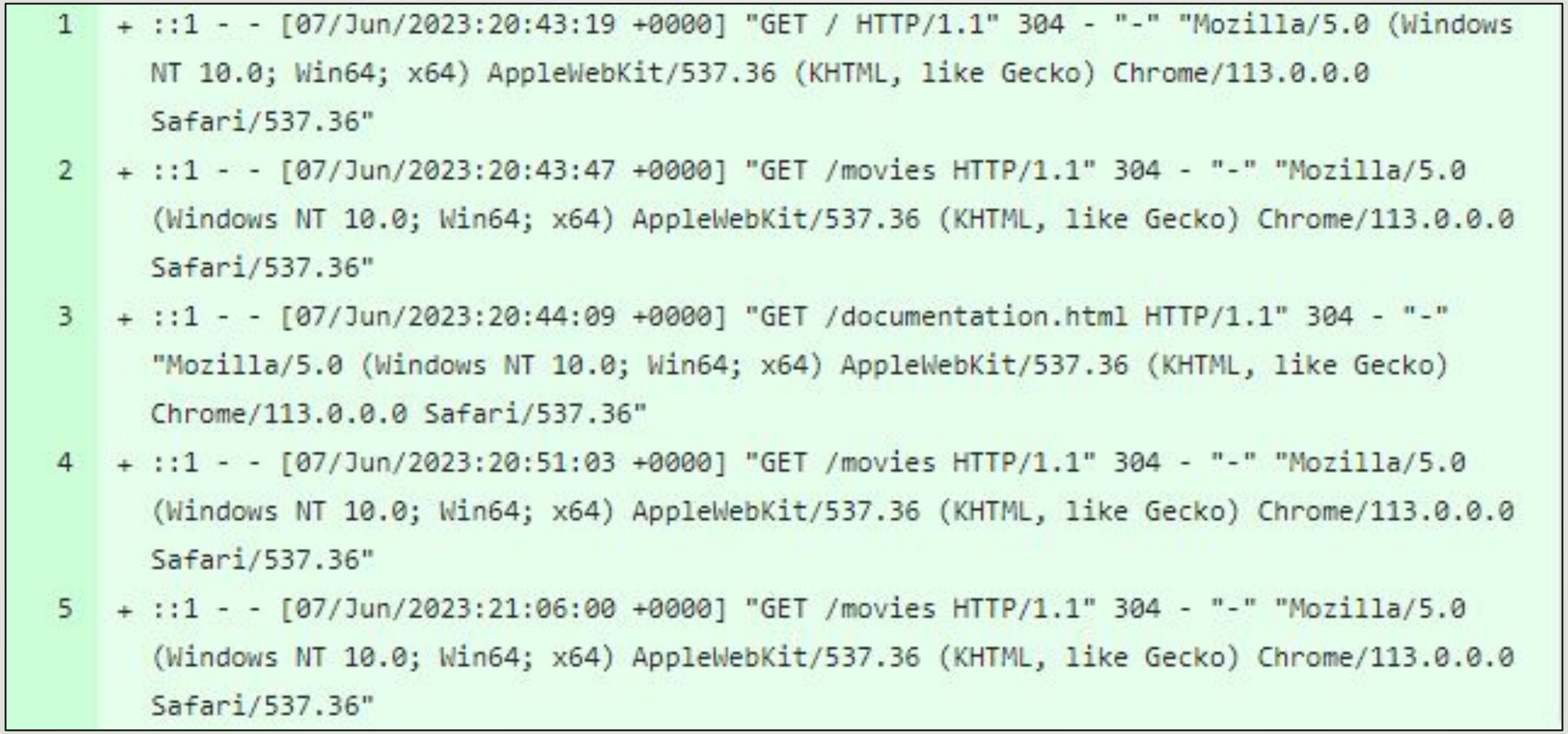
## **WHAT** WAS THE GOAL

Importing and combining the HTTP, URL and FS modules inside a newly created server.js document.

Putting these modules into work by parsing incoming URL test requests to determine if they contain a specific word. If the verification was true, “documentation.html” file was meant to be returned to the user having sent the request, otherwise “index.html” was returned.

Regarding the FS module, I’ve created a txt.log file and I ensured that for all requests made, this module was used to log both the request URL and a timestamp inside the log.txt file.

## **SAMPLE** OF REQUESTS LOGGED IN A LOG.TXT FILE USING FS MODULE



```
+ ::1 - - [07/Jun/2023:20:43:19 +0000] "GET / HTTP/1.1" 304 - "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/113.0.0.0 Safari/537.36"
+ ::1 - - [07/Jun/2023:20:43:47 +0000] "GET /movies HTTP/1.1" 304 - "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/113.0.0.0 Safari/537.36"
+ ::1 - - [07/Jun/2023:20:44:09 +0000] "GET /documentation.html HTTP/1.1" 304 - "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/113.0.0.0 Safari/537.36"
+ ::1 - - [07/Jun/2023:20:51:03 +0000] "GET /movies HTTP/1.1" 304 - "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/113.0.0.0 Safari/537.36"
+ ::1 - - [07/Jun/2023:21:06:00 +0000] "GET /movies HTTP/1.1" 304 - "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/113.0.0.0 Safari/537.36"
```

## **CHALLENGES** OR SPECIAL POINTS OF CONSIDERATION

I was in my first experimentation with the use of Node.js in general, as well as Node built-in modules. It was therefore a good challenge to understand the key concepts of how this environment works at first, but I was able to use several solutions by myself to assimilate everything and eventually deliver all expected requirements. Some of the solutions I used to face the challenges:

- Doing several researches on various platforms
- Analyzing other codes / projects using same logics
- Carrying out multiple tests locally and try different potential solutions

## 03

**Getting to know** the Node package manager npm and **installing** the necessary packages / third party modules for the API

Skills used

Research

## **WHY** WAS THIS STEP IMPORTANT

Node package managers are essential to automatically check and ensure all the project packages / dependencies are up-to-date. Knowing how to use a node package manager is therefore important to save time and prevent issues that may arise in the app from out-of-date packages / dependencies (whether as global, local or development dependencies). This also gave me the opportunity to learn about Semantic Versioning (SemVer).

Regarding the packages to be installed, it was important to ensure that they would be available to use over the next step when starting to implement more codes in the files.

## **WHAT** WAS THE GOAL

Installing the key packages needed in order to build the API (Express and Body-parser). Creating an index.js file and initializing the project with a package.json file (npm init).

**Using** Express framework and **implementing** an error handler

Skills used

Research
Code writing

## **WHY** WAS THIS STEP IMPORTANT

Using Express framework allows for a quicker, shorter and more concise way of creating APIs and sending HTTP requests back and forth between frontend and backend, thus increasing work efficiency and time required to create backend logic. As for the error handler, it was important to create a code that could handle unanticipated errors and work as a net/alert if something unexpectedly was to break in the code.

As for the error handler, it was important to create a code that could handle unanticipated errors and work as a net/alert if something unexpected was to break in the code.

## **WHAT** WAS THE GOAL

Installing Express framework using npm and importing it inside index.js file. Refactoring previous codes using Express framework to handle some of the same functionalities. Using Express routing syntax to create two new routes:

- An Express GET route located at the endpoint “/movies” (that endpoint returns a JSON object containing data about all movies)
- An Express GET route located at the endpoint “/” that returns a custom default message (Welcome to your new movie e-friend advisor!)

Using the Morgan middleware library to log all requests (instead of using the FS module to write down requests in a log.txt file like previously set up).

Creating an error-handling middleware function that logs all application-level errors in the terminal.

```
$ npm install morgan
```

```
const  morgan = require('morgan');
```

## **CHALLENGES** OR SPECIAL POINTS OF CONSIDERATION

Getting used to the logic behind the Express framework took me some time and some additional readings. I wanted to make sure I understand well the logic behind Express since this framework was going to be the basis of all my backend structure. I also wanted to ensure I understood the added value of using Express rather than using other methods with similar results.

## 05

GET

PUT

POST

DELETE

**Looking** into RESTful architecture, **creating** multiple routes for my REST API and **documenting** it

### Skills used

Research
Code writing

## **WHY** WAS THIS STEP IMPORTANT

REST API allows to send HTTP requests (GET, PUT, POST, and DELETE) to my web server, which are then translated to appropriate CRUD operations on the server's data. This architecture was important to implement in the backend logic to allow users to access data from my web server/database and interact with it in different ways.

Documenting how the API is built is also important since it informs client applications on how to format requests to the API, such as what URL endpoints to target or what data to send as parameters and what to expect as responses from it.

## **WHAT** WAS THE GOAL

Defining what data I wanted my server to expose by considering key questions such as: what information could the client want to retrieve (or GET) from the server? What information could it want to add (or POST)? To update (PUT)? To remove (DELETE)?

Thinking about the general architecture of my API, as well as the structure of the endpoints to be created. Associating each of these endpoints with a specific operation following the project requirements and creating the Express codes to route all of the endpoints.

```
app.get('/')
app.get('/movies'
app.get('/movies/:Title')
app.get('/movies/genre/:genreName')
app.get('/movies/directors/:directorName')
app.post('/users')
app.put('/users/:Username')
app.post('/users/:Username/movies/:MovieID')
app.delete('/users/:Username/movies/:MovieID')
app.delete('/users/:Username')
```

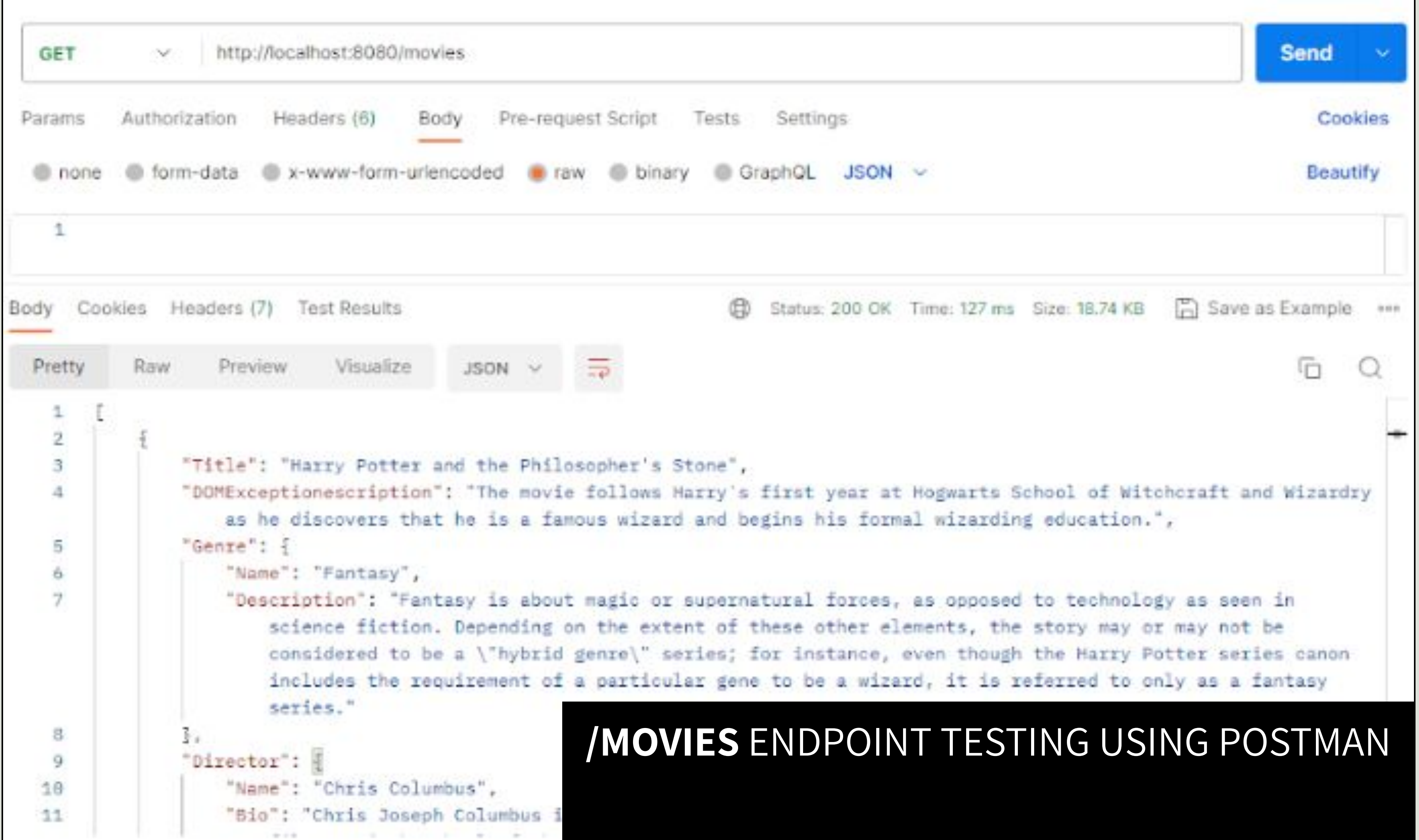
## **WHAT** WAS THE GOAL (SUITE)

Installing a Uuid module to assign an ID to any newly created object, such as a new user account (and so eliminate the need for users to come up with an ID themselves when creating their profile, and ensures that there are never two of the same IDs for objects).

Testing my API and newly created endpoints with Postman (a tool helping with API development - see sample pictures in the next slides). To run those tests, the data I used was stored in-memory in a JS testing file, but it was planned to eventually build and use an external database over the next steps (see step 7) to store movie and user information.

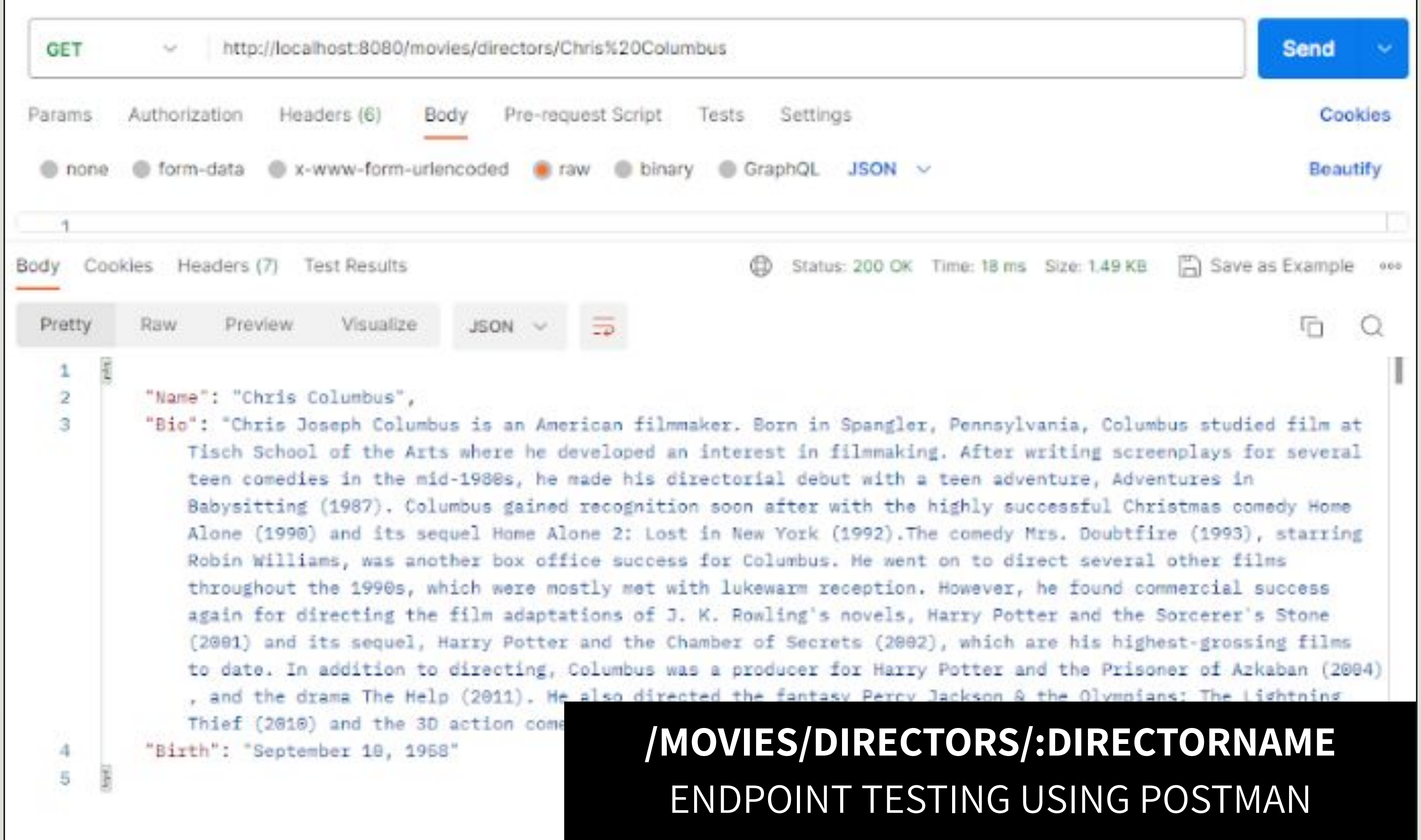
Documenting the API endpoints to ensure that developers of the client app know what data they need to send along with their requests, as well as what kind of data they can expect back in response.

POSTMAN



```
[
    {
        "Title": "Harry Potter and the Philosopher's Stone",
        "DOMExceptionescription": "The movie follows Harry's first year at Hogwarts School of Witchcraft and Wizardry
            as he discovers that he is a famous wizard and begins his formal wizarding education.",
        "Genre": {
            "Name": "Fantasy",
            "Description": "Fantasy is about magic or supernatural forces, as opposed to technology as seen in
                science fiction. Depending on the extent of these other elements, the story may or may not be
                considered to be a \"hybrid genre\" series; for instance, even though the Harry Potter series canon
                includes the requirement of a particular gene to be a wizard, it is referred to only as a fantasy
                series."
        },
        "Director": {
            "Name": "Chris Columbus",
            "Bio": "Chris Joseph Columbus i
```

## **/MOVIES** ENDPOINT TESTING USING POSTMAN



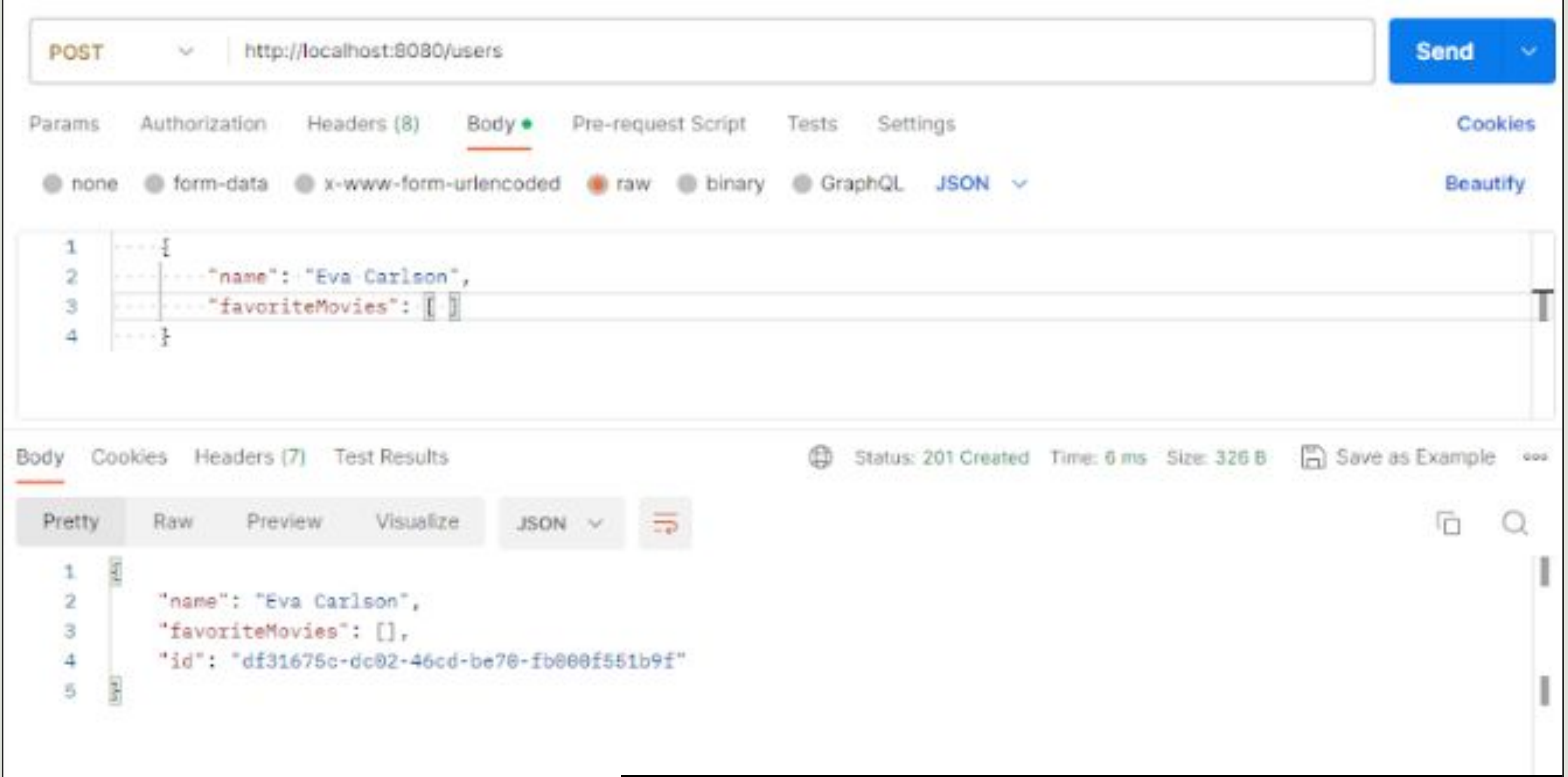
```
http://localhost:8080/movies/directors/Chris%20Columbus
```

Status: 200 OK Time: 18 ms Size: 1.49 KB

```
{
    "Name": "Chris Columbus",
    "Bio": "Chris Joseph Columbus is an American filmmaker. Born in Spangler, Pennsylvania, Columbus studied film at
        Tisch School of the Arts where he developed an interest in filmmaking. After writing screenplays for several
        teen comedies in the mid-1980s, he made his directorial debut with a teen adventure, Adventures in
        Babysitting (1987). Columbus gained recognition soon after with the highly successful Christmas comedy Home
        Alone (1990) and its sequel Home Alone 2: Lost in New York (1992).The comedy Mrs. Doubtfire (1993), starring
        Robin Williams, was another box office success for Columbus. He went on to direct several other films
        throughout the 1990s, which were mostly met with lukewarm reception. However, he found commercial success
        again for directing the film adaptations of J. K. Rowling's novels, Harry Potter and the Sorcerer's Stone
        (2001) and its sequel, Harry Potter and the Chamber of Secrets (2002), which are his highest-grossing films
        to date. In addition to directing, Columbus was a producer for Harry Potter and the Prisoner of Azkaban (2004)
        , and the drama The Help (2011). He also directed the fantasy Percy Jackson & the Olympians: The Lightning
        Thief (2010) and the 3D action come
    "Birth": "September 10, 1958"
}
```

## /MOVIES/DIRECTORS/:DIRECTORNAME

ENDPOINT TESTING USING POSTMAN



```
{
    "name": "Eva Carlson",
    "favoriteMovies": [ ]
}
```

```
{
    "name": "Eva Carlson",
    "favoriteMovies": [],
    "id": "df31675c-dc02-46cd-be70-fb000f551b9f"
}
```

## **/USERS** ENDPOINT TESTING USING POSTMAN

## **CHALLENGES** OR SPECIAL POINTS OF CONSIDERATION

I was in my first experiences with the RESTful API architecture. Many new elements were therefore to be understood. However, with the right amount of time and efforts, I made it up to understand the logic behind RESTful API and some modules used at this stage (ex: Body-parser and Uuid). Being able to test my codes with Postman facilitated my understanding of the logic behind the creation of each endpoint, both at the request level and the response returned.

## **DECISIONS** MADE

I defined the endpoints for my REST API according to project requirement, and documented them in a README.md more easily accessible on Github, and in a JSDoc).

06

**Creating** a relational database (SQL) using PostgreSQL

Skills used

Research

## **WHY** WAS THIS STEP IMPORTANT

This step was mainly meant to learn the logic of a relational database, but not to actually use one in the project since it was planned that myFlix would use a non-relational database (MongoDB). However, since SQL and relational databases remain important in different working / project environments, it was still important to get some practice in creating and manipulating data in a relational database.

Moreover, even if a relational database was not to be used in the project, the raw data collected in this step have been re-used to set up my non-relational database in the next step, making this still contributing to the whole web app development.

# **WHAT** WAS THE GOAL

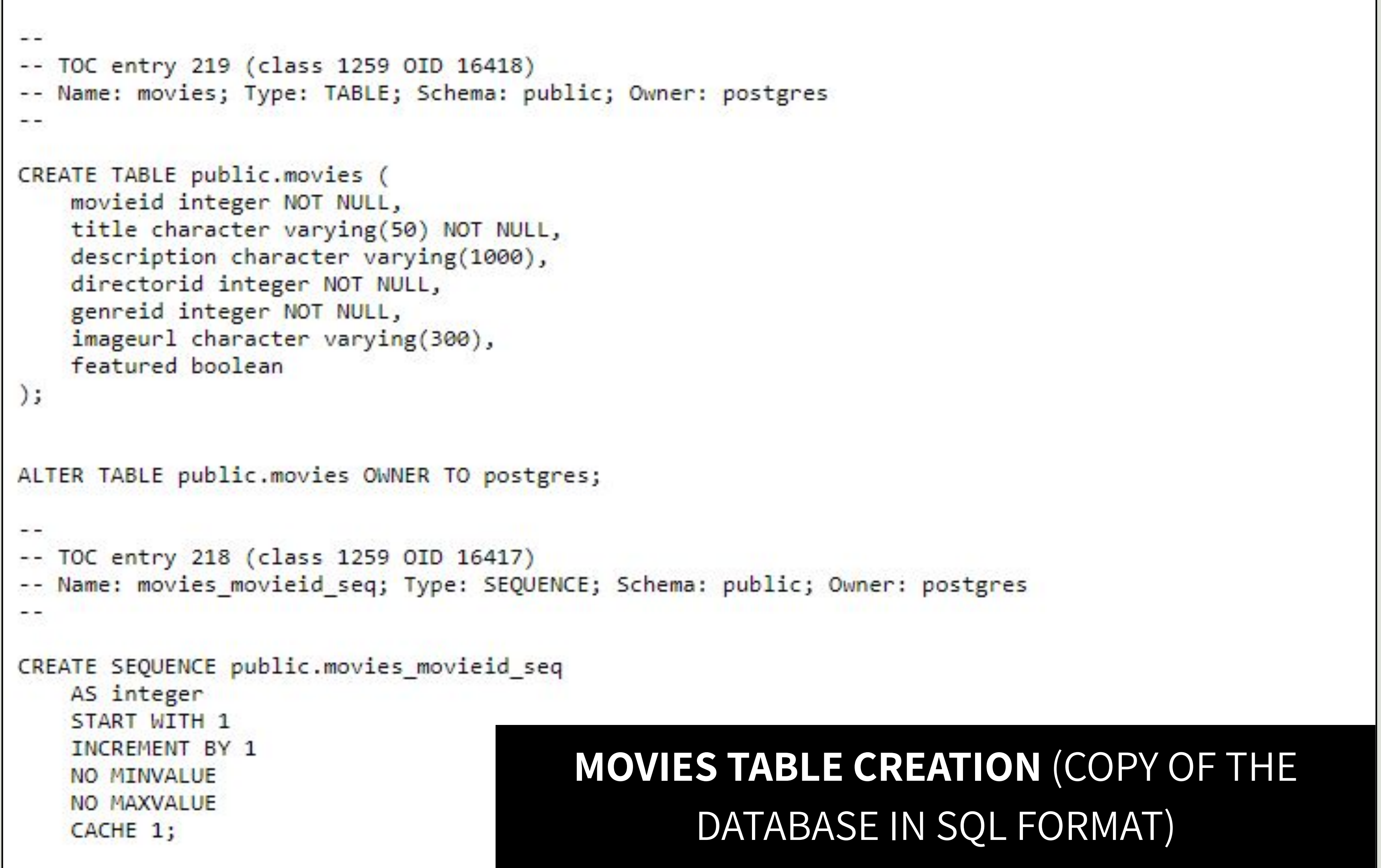
Learning the basics of designing a relational database (schemas, entities and attributes, foreign keys, junction tables, etc.). Defining the entities of my web app and their corresponding attributes. For example:

- Movies as entity - ID, Title, Description, Genre, Directors and more as attributes
- Users as entity - ID, Username, Password, Email, Birthday and Favorite movies as attributes

Defining how all this information is related / what are the relationships between my entities and attributes (one-to-one, one-to-many, many-to-many).

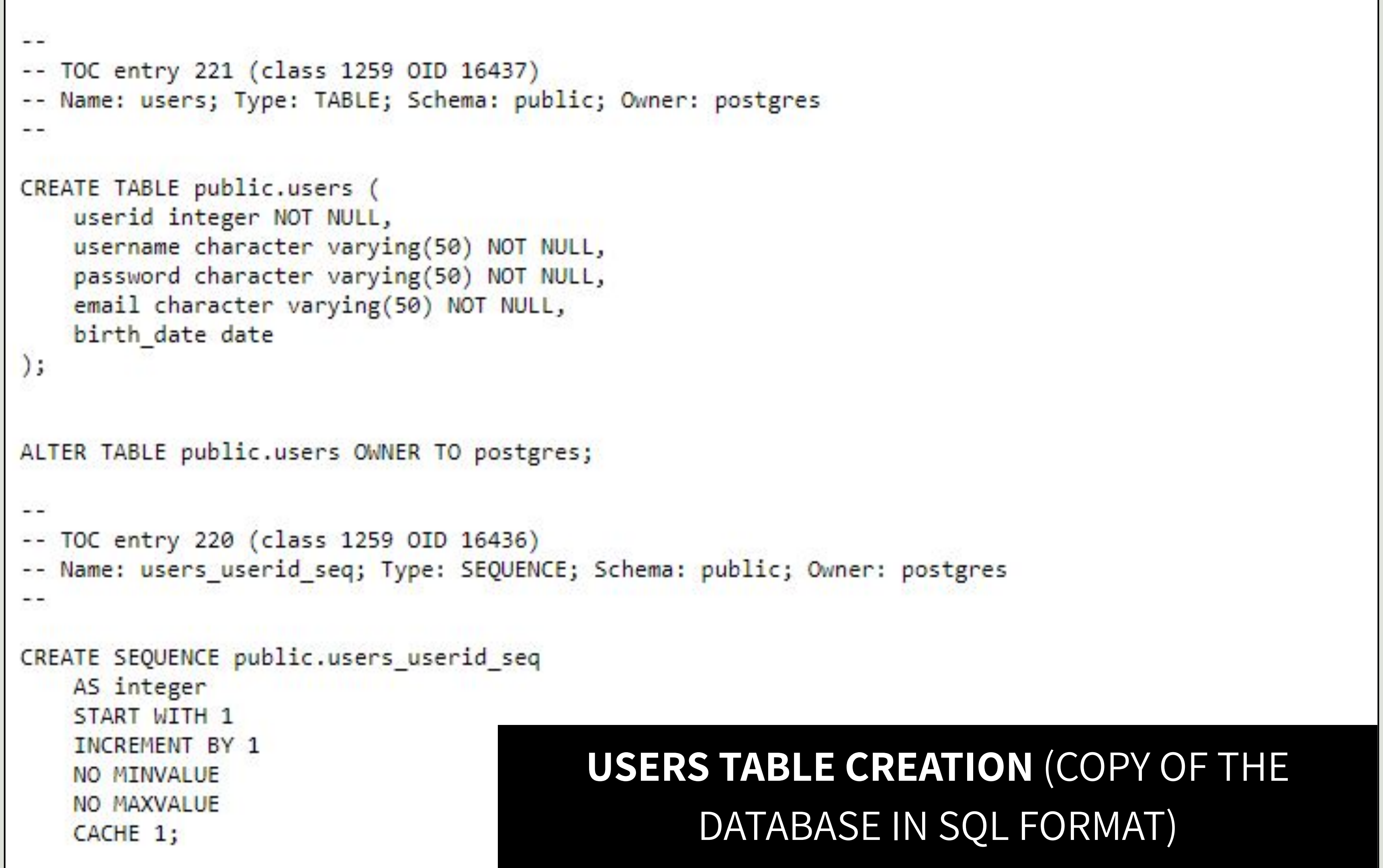
Creating a database in the relational database management systems (RDBMS) PostgreSQL and creating different tables (for Movies, Genres, Directors, Users and Users-Movies) inside of it (see pictures in the next slides). Populating each of these tables with some information using SQL language.

Learning and practicing querying / performing CRUD operations on the data stored within the tables of the database using SQL language SELECT, UPDATE and DELETE (see pictures in the next slides).



```
--
-- TOC entry 219 (class 1259 OID 16418)
-- Name: movies; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.movies (
    movieid integer NOT NULL,
    title character varying(50) NOT NULL,
    description character varying(1000),
    directorid integer NOT NULL,
    genreid integer NOT NULL,
    imageurl character varying(300),
    featured boolean
);


ALTER TABLE public.movies OWNER TO postgres;

--
-- TOC entry 218 (class 1259 OID 16417)
-- Name: movies_movieid_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

CREATE SEQUENCE public.movies_movieid_seq
    AS integer
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;
```

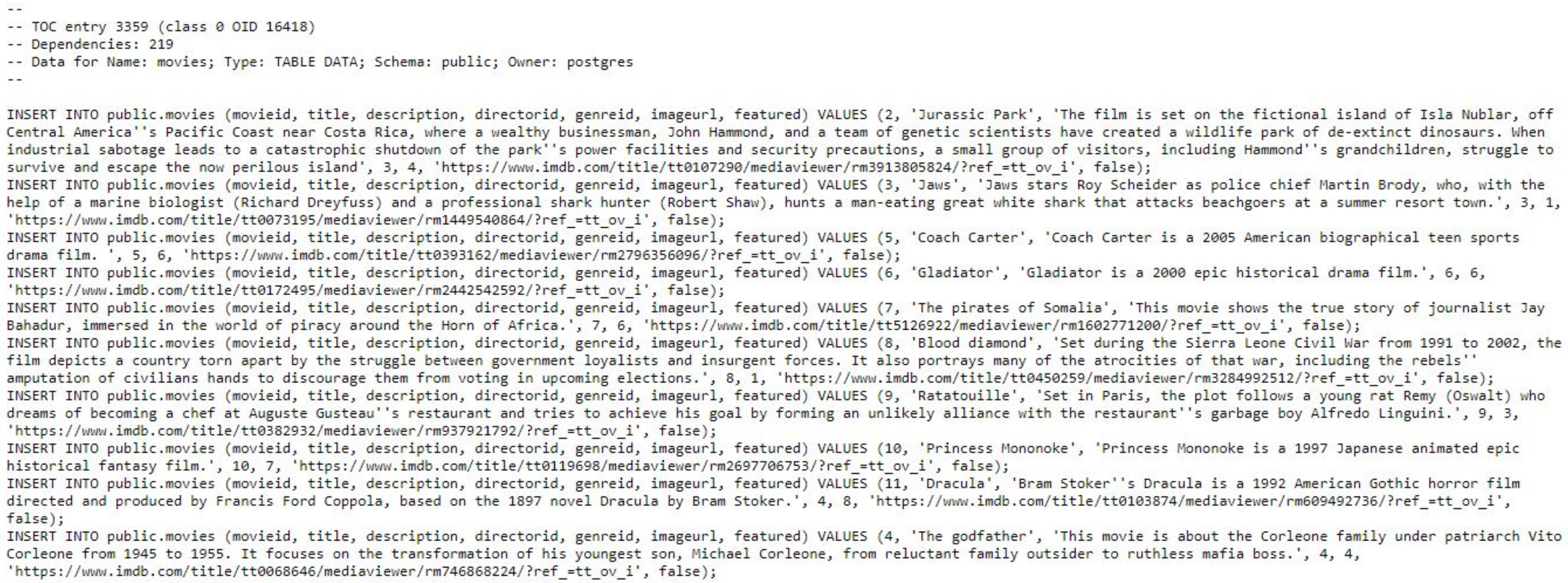
## **MOVIES TABLE CREATION** (COPY OF THE DATABASE IN SQL FORMAT)



```
--
-- TOC entry 221 (class 1259 OID 16437)
-- Name: users; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.users (
    userid integer NOT NULL,
    username character varying(50) NOT NULL,
    password character varying(50) NOT NULL,
    email character varying(50) NOT NULL,
    birth_date date
);


ALTER TABLE public.users OWNER TO postgres;

--
-- TOC entry 220 (class 1259 OID 16436)
-- Name: users_userid_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

CREATE SEQUENCE public.users_userid_seq
    AS integer
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;
```

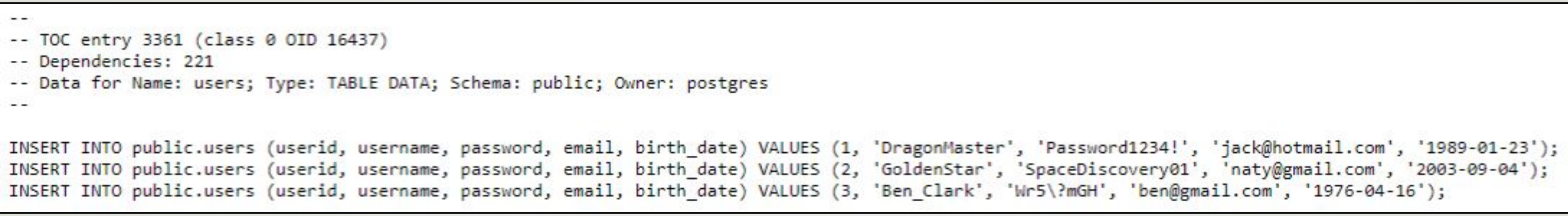
**USERS TABLE CREATION** (COPY OF THE DATABASE IN SQL FORMAT)

# **POPULATING** MOVIES TABLE WITH DATA (COPY OF THE DATABASE IN SQL FORMAT)



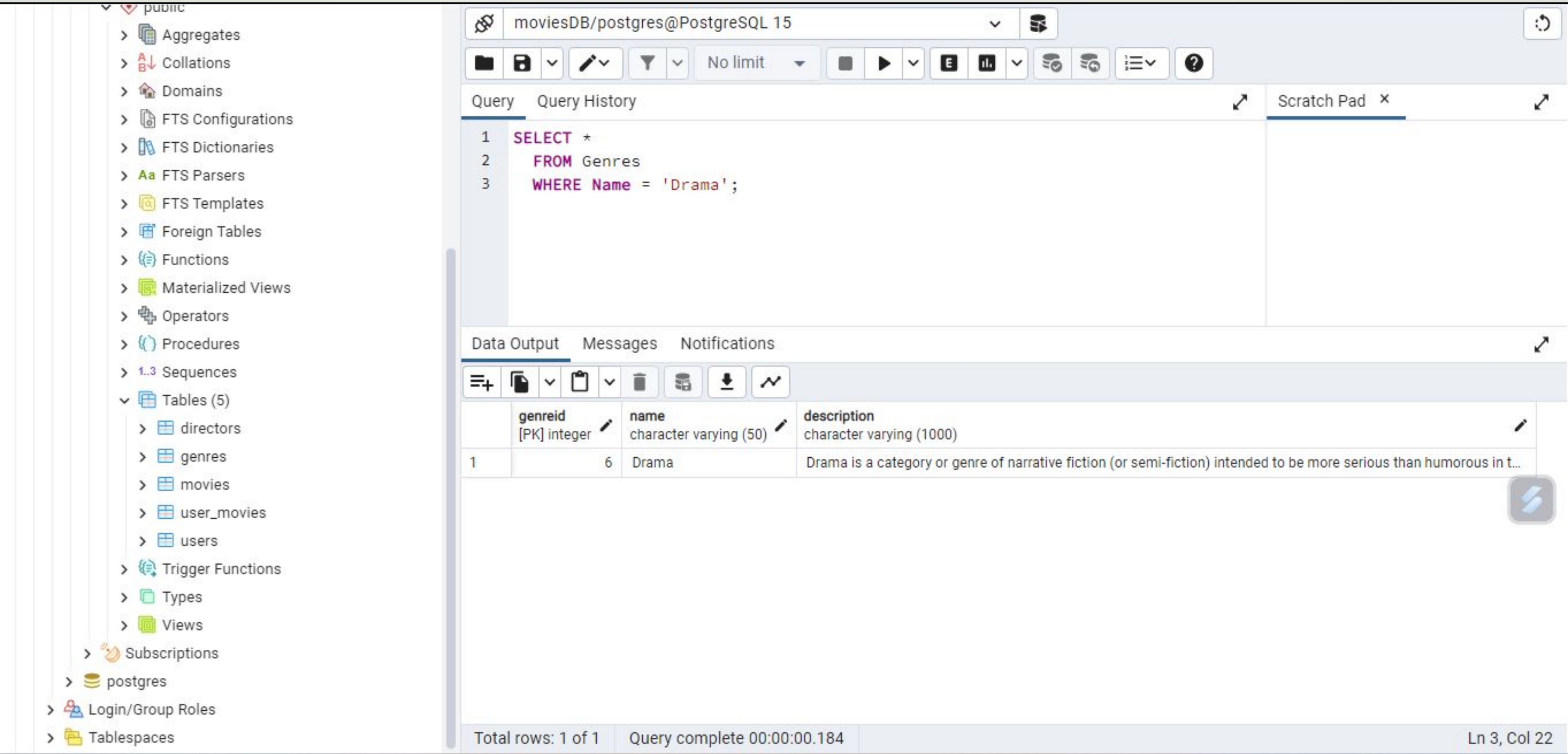
```
--
-- TOC entry 3359 (class 0 OID 16418)
-- Dependencies: 219
-- Data for Name: movies; Type: TABLE DATA; Schema: public; Owner: postgres
--

INSERT INTO public.movies (movieid, title, description, directorid, genreid, imageurl, featured) VALUES (2, 'Jurassic Park', 'The film is set on the fictional island of Isla Nublar, off
Central America''s Pacific Coast near Costa Rica, where a wealthy businessman, John Hammond, and a team of genetic scientists have created a wildlife park of de-extinct dinosaurs. When
industrial sabotage leads to a catastrophic shutdown of the park''s power facilities and security precautions, a small group of visitors, including Hammond''s grandchildren, struggle to
survive and escape the now perilous island', 3, 4, 'https://www.imdb.com/title/tt0107290/mediaviewer/rm3913805824/?ref_=tt_ov_i', false);
INSERT INTO public.movies (movieid, title, description, directorid, genreid, imageurl, featured) VALUES (3, 'Jaws', 'Jaws stars Roy Scheider as police chief Martin Brody, who, with the
help of a marine biologist (Richard Dreyfuss) and a professional shark hunter (Robert Shaw), hunts a man-eating great white shark that attacks beachgoers at a summer resort town.', 3, 1,
'https://www.imdb.com/title/tt0073195/mediaviewer/rm1449540864/?ref_=tt_ov_i', false);
INSERT INTO public.movies (movieid, title, description, directorid, genreid, imageurl, featured) VALUES (5, 'Coach Carter', 'Coach Carter is a 2005 American biographical teen sports
drama film. ', 5, 6, 'https://www.imdb.com/title/tt0393162/mediaviewer/rm2796356096/?ref_=tt_ov_i', false);
INSERT INTO public.movies (movieid, title, description, directorid, genreid, imageurl, featured) VALUES (6, 'Gladiator', 'Gladiator is a 2000 epic historical drama film.', 6, 6,
'https://www.imdb.com/title/tt0172495/mediaviewer/rm2442542592/?ref_=tt_ov_i', false);
INSERT INTO public.movies (movieid, title, description, directorid, genreid, imageurl, featured) VALUES (7, 'The pirates of Somalia', 'This movie shows the true story of journalist Jay
Bahadur, immersed in the world of piracy around the Horn of Africa.', 7, 6, 'https://www.imdb.com/title/tt5126922/mediaviewer/rm1602771200/?ref_=tt_ov_i', false);
INSERT INTO public.movies (movieid, title, description, directorid, genreid, imageurl, featured) VALUES (8, 'Blood diamond', 'Set during the Sierra Leone Civil War from 1991 to 2002, the
film depicts a country torn apart by the struggle between government loyalists and insurgent forces. It also portrays many of the atrocities of that war, including the rebels''
amputation of civilians hands to discourage them from voting in upcoming elections.', 8, 1, 'https://www.imdb.com/title/tt0450259/mediaviewer/rm3284992512/?ref_=tt_ov_i', false);
INSERT INTO public.movies (movieid, title, description, directorid, genreid, imageurl, featured) VALUES (9, 'Ratatouille', 'Set in Paris, the plot follows a young rat Remy (Oswalt) who
dreams of becoming a chef at Auguste Gusteau''s restaurant and tries to achieve his goal by forming an unlikely alliance with the restaurant''s garbage boy Alfredo Linguini.', 9, 3,
'https://www.imdb.com/title/tt0382932/mediaviewer/rm937921792/?ref_=tt_ov_i', false);
INSERT INTO public.movies (movieid, title, description, directorid, genreid, imageurl, featured) VALUES (10, 'Princess Mononoke', 'Princess Mononoke is a 1997 Japanese animated epic
historical fantasy film.', 10, 7, 'https://www.imdb.com/title/tt0119698/mediaviewer/rm2697706753/?ref_=tt_ov_i', false);
INSERT INTO public.movies (movieid, title, description, directorid, genreid, imageurl, featured) VALUES (11, 'Dracula', 'Bram Stoker''s Dracula is a 1992 American Gothic horror film
directed and produced by Francis Ford Coppola, based on the 1897 novel Dracula by Bram Stoker.', 4, 8, 'https://www.imdb.com/title/tt0103874/mediaviewer/rm609492736/?ref_=tt_ov_i',
false);
INSERT INTO public.movies (movieid, title, description, directorid, genreid, imageurl, featured) VALUES (4, 'The godfather', 'This movie is about the Corleone family under patriarch Vito
Corleone from 1945 to 1955. It focuses on the transformation of his youngest son, Michael Corleone, from reluctant family outsider to ruthless mafia boss.', 4, 4,
'https://www.imdb.com/title/tt0068646/mediaviewer/rm746868224/?ref_=tt_ov_i', false);
```

# **POPULATING** USERS TABLE WITH DATA (COPY OF THE DATABASE IN SQL FORMAT)



```
--
-- TOC entry 3361 (class 0 OID 16437)
-- Dependencies: 221
-- Data for Name: users; Type: TABLE DATA; Schema: public; Owner: postgres
--

INSERT INTO public.users (userid, username, password, email, birth_date) VALUES (1, 'DragonMaster', 'Password1234!', 'jack@hotmail.com', '1989-01-23');
INSERT INTO public.users (userid, username, password, email, birth_date) VALUES (2, 'GoldenStar', 'SpaceDiscovery01', 'naty@gmail.com', '2003-09-04');
INSERT INTO public.users (userid, username, password, email, birth_date) VALUES (3, 'Ben_Clark', 'Wr5\?mGH', 'ben@gmail.com', '1976-04-16');
```

# **SELECT** OPERATION (CRUD - READ)

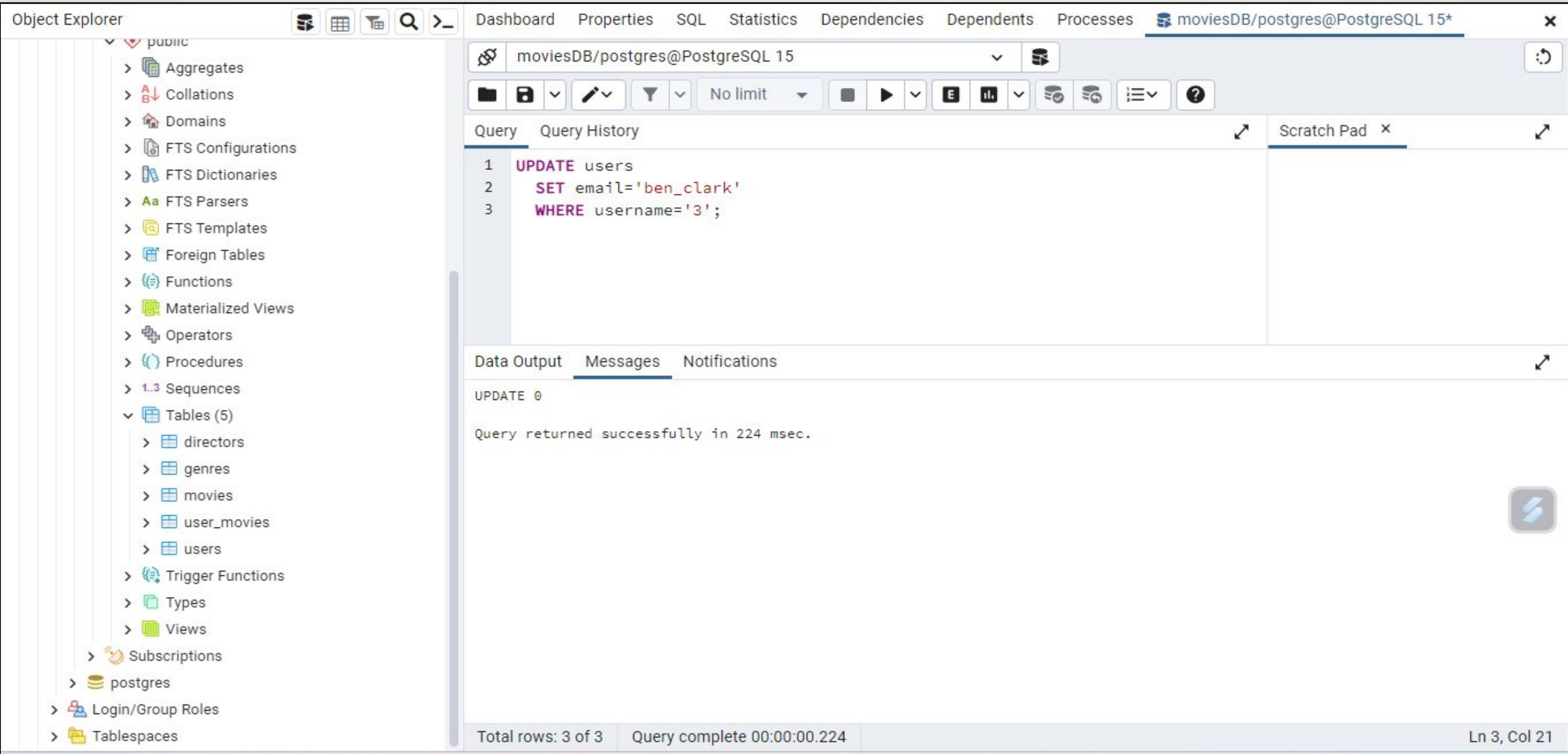


```sql
SELECT *
  FROM Genres
  WHERE Name = 'Drama';
```

| | genreid [PK] integer | name character varying (50) | description character varying (1000) |
|---|---|---|---|
| 1 | 6 | Drama | Drama is a category or genre of narrative fiction (or semi-fiction) intended to be more serious than humorous in t... |

Total rows: 1 of 1    Query complete 00:00:00.184    Ln 3, Col 22

# **UPDATE** OPERATION (CRUD - UPDATE)

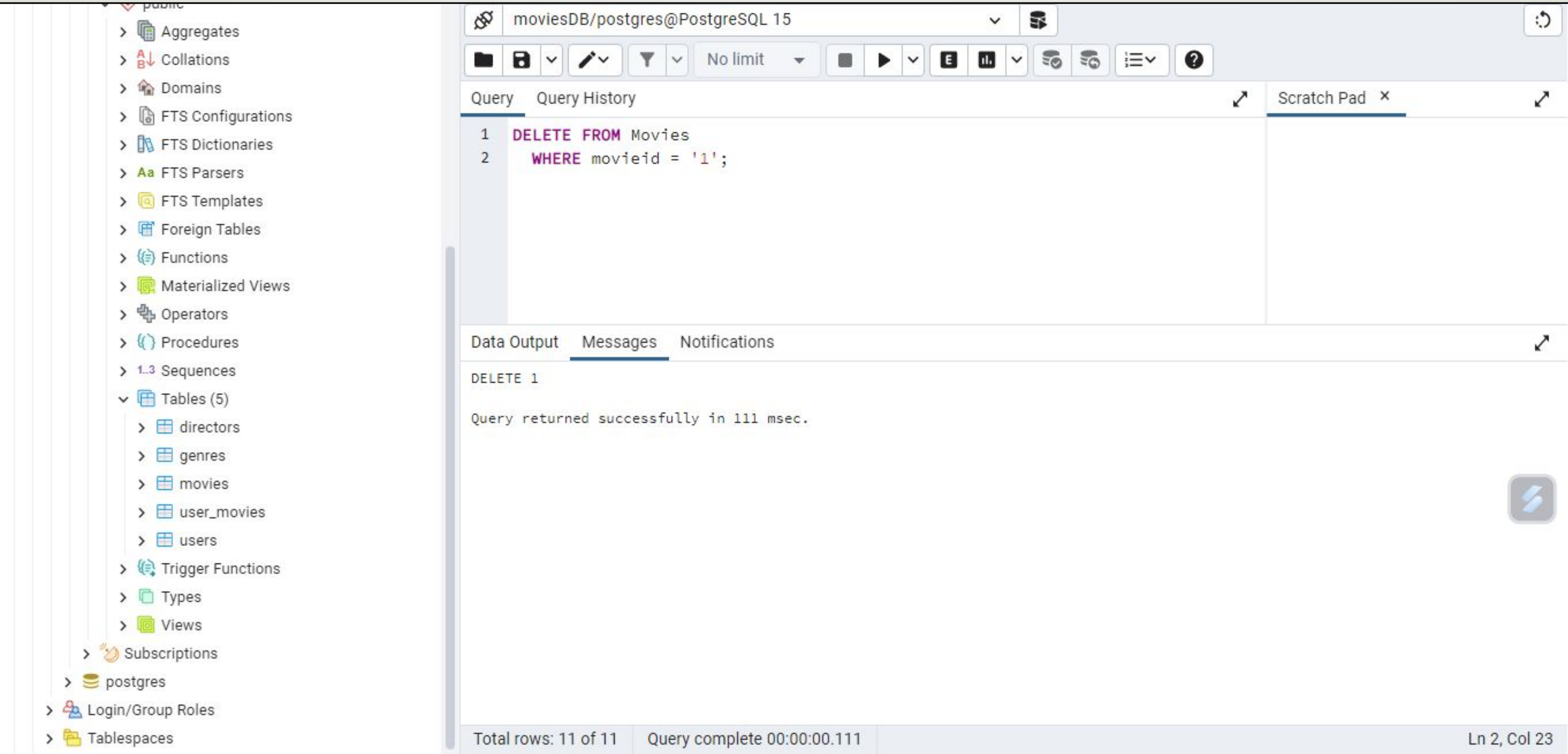


```sql
UPDATE users
  SET email='ben_clark'
  WHERE username='3';
```

```
UPDATE 0

Query returned successfully in 224 msec.
```

Total rows: 3 of 3 Query complete 00:00:00.224 Ln 3, Col 21

# **DELETE** OPERATION (CRUD - DELETE)



```sql
DELETE FROM Movies
  WHERE movieid = '1';
```

```
DELETE 1

Query returned successfully in 111 msec.
```

Total rows: 11 of 11    Query complete 00:00:00.111    Ln 2, Col 23

## **CHALLENGES** OR SPECIAL POINTS OF CONSIDERATION

Since I already had some experience working with relational databases within ArcGIS (GIS software), setting up and understanding the logic and interface of a relational database and querying it was not too difficult

## **DECISIONS** MADE

Deciding the sort of data to be stored in the database (necessary for this step in the conception of the relational database, but also necessary and important at this point since these data would be re-used in the non-relational database built in the next step).

## 07

**Creating** a non-relational database (NoSQL) using MongoDB and querying it using CRUD operations

Skills used

Research
Code writing
Debugging

## **WHY** WAS THIS STEP IMPORTANT

Creating an external database for my REST API to interact with and ensure all CRUD operations are fulfilled was important, since up to this point, the data used were stored in-memory in a JavaScript file for testing purposes.

READ

CREATE

DELETE

UPDATE

## **WHAT** WAS THE GOAL

Getting to know the different models of NoSQL databases (Key-Value Stores, Document-Based Stores, Graph Stores and Wide-Column Stores), and more specifically the Document-Based Stores as this is the MongoDB type (MongoDB being part of the MERN tech stack used for this project).

Understanding the logic behind MongoDB data structure (collections and documents, key-value pairs structure within documents, embedded documents, references to create links between documents in different collections) and how to interact with it using JavaScript.

## **WHAT** WAS THE GOAL (SUITE)

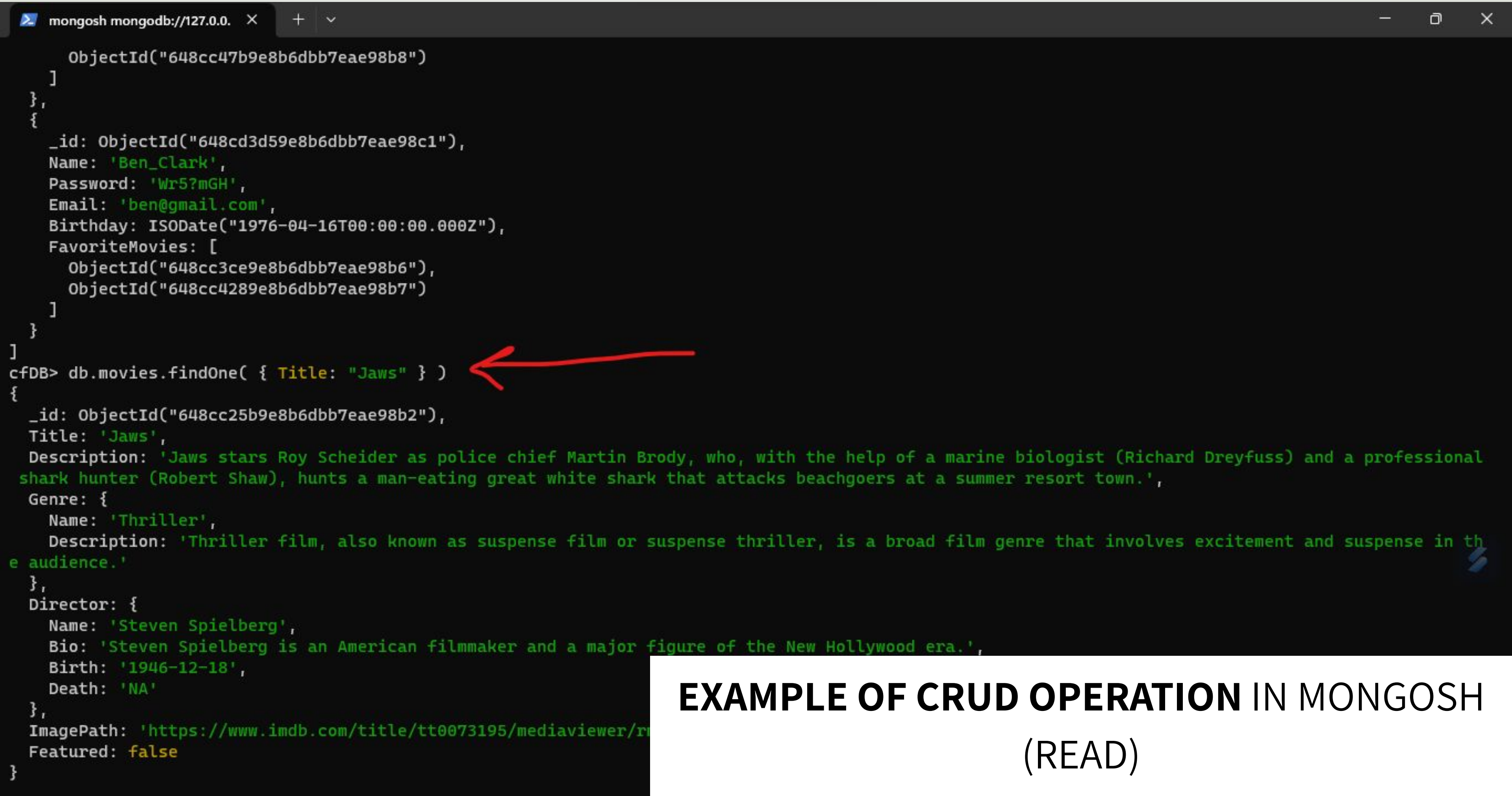
Installing MongoDB, including MongoDB Community Server, MongoDB Shell (Mongosh) and MongoDB Database Tools.

Creating a local non-relational database to feed my web app (rather than feeding it with in-memory data stored in a file like it was the case before this step). Populate my users and movies collections, using embedded documents for some specific movie information inside the movie collection.

Querying my local database to test it out using the four CRUD operations (via Mongo Shell - Mongosh).

## **CHALLENGES** OR SPECIAL POINTS OF CONSIDERATION

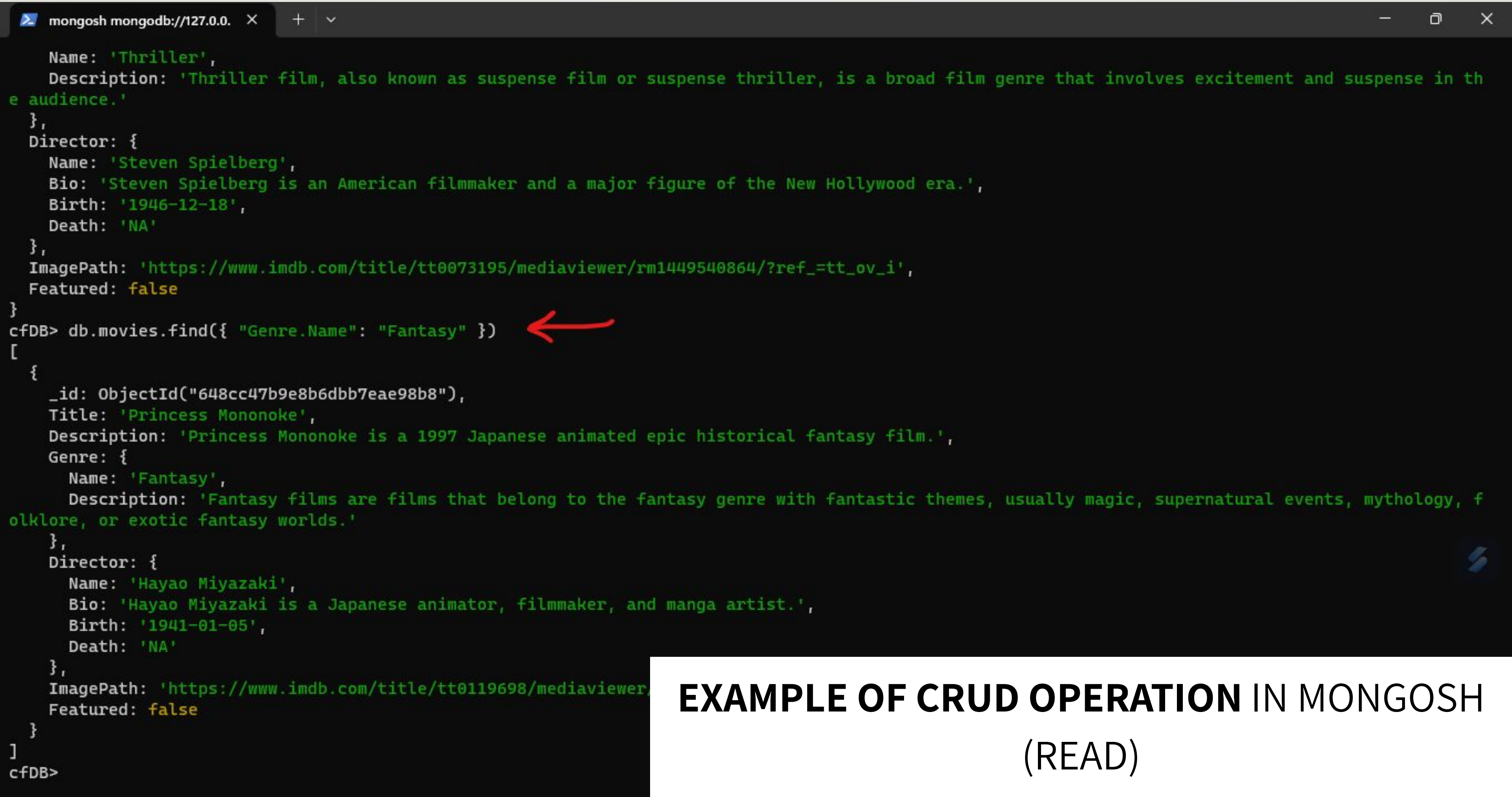
At first, I had problems launching and using MongoDB Shell - Mongosh. I was always receiving an error message when trying to launch it in my terminal. I later found that the issue was lying in the way I've installed it. I therefore un-installed Mongosh and installed it again, but this time using other parameters in the installation process, which fixed the issue.



mongosh mongodb://127.0.0.

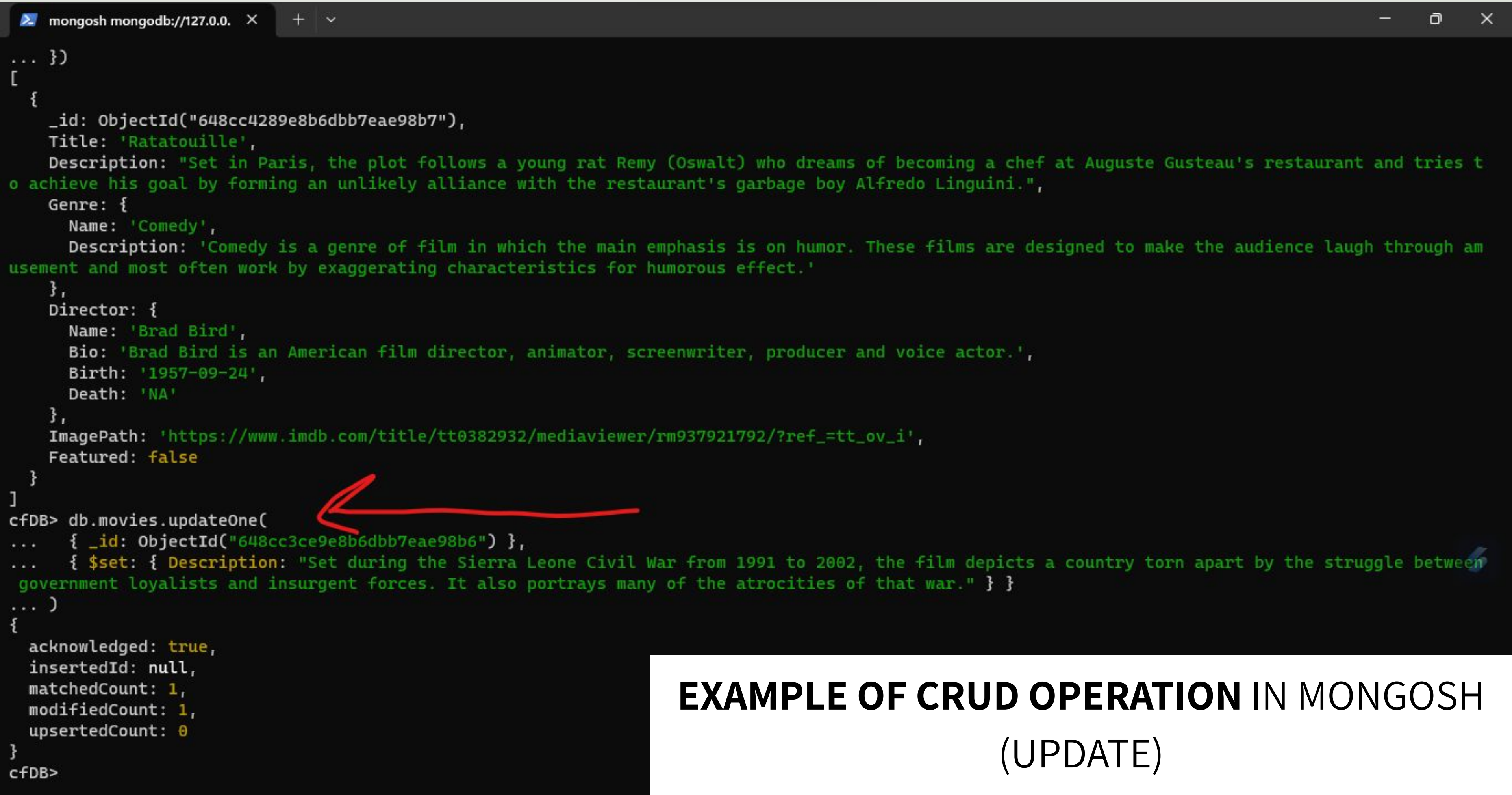
```
      ObjectId("648cc47b9e8b6dbb7eae98b8")
    ]
  },
  {
    _id: ObjectId("648cd3d59e8b6dbb7eae98c1"),
    Name: 'Ben_Clark',
    Password: 'Wr5?mGH',
    Email: 'ben@gmail.com',
    Birthday: ISODate("1976-04-16T00:00:00.000Z"),
    FavoriteMovies: [
      ObjectId("648cc3ce9e8b6dbb7eae98b6"),
      ObjectId("648cc4289e8b6dbb7eae98b7")
    ]
  }
]
cfDB> db.movies.findOne( { Title: "Jaws" } )
{
  _id: ObjectId("648cc25b9e8b6dbb7eae98b2"),
  Title: 'Jaws',
  Description: 'Jaws stars Roy Scheider as police chief Martin Brody, who, with the help of a marine biologist (Richard Dreyfuss) and a professional
 shark hunter (Robert Shaw), hunts a man-eating great white shark that attacks beachgoers at a summer resort town.',
  Genre: {
    Name: 'Thriller',
    Description: 'Thriller film, also known as suspense film or suspense thriller, is a broad film genre that involves excitement and suspense in th
e audience.'
  },
  Director: {
    Name: 'Steven Spielberg',
    Bio: 'Steven Spielberg is an American filmmaker and a major figure of the New Hollywood era.',
    Birth: '1946-12-18',
    Death: 'NA'
  },
  ImagePath: 'https://www.imdb.com/title/tt0073195/mediaviewer/r
  Featured: false
}
```

**EXAMPLE OF CRUD OPERATION** IN MONGOSH (READ)



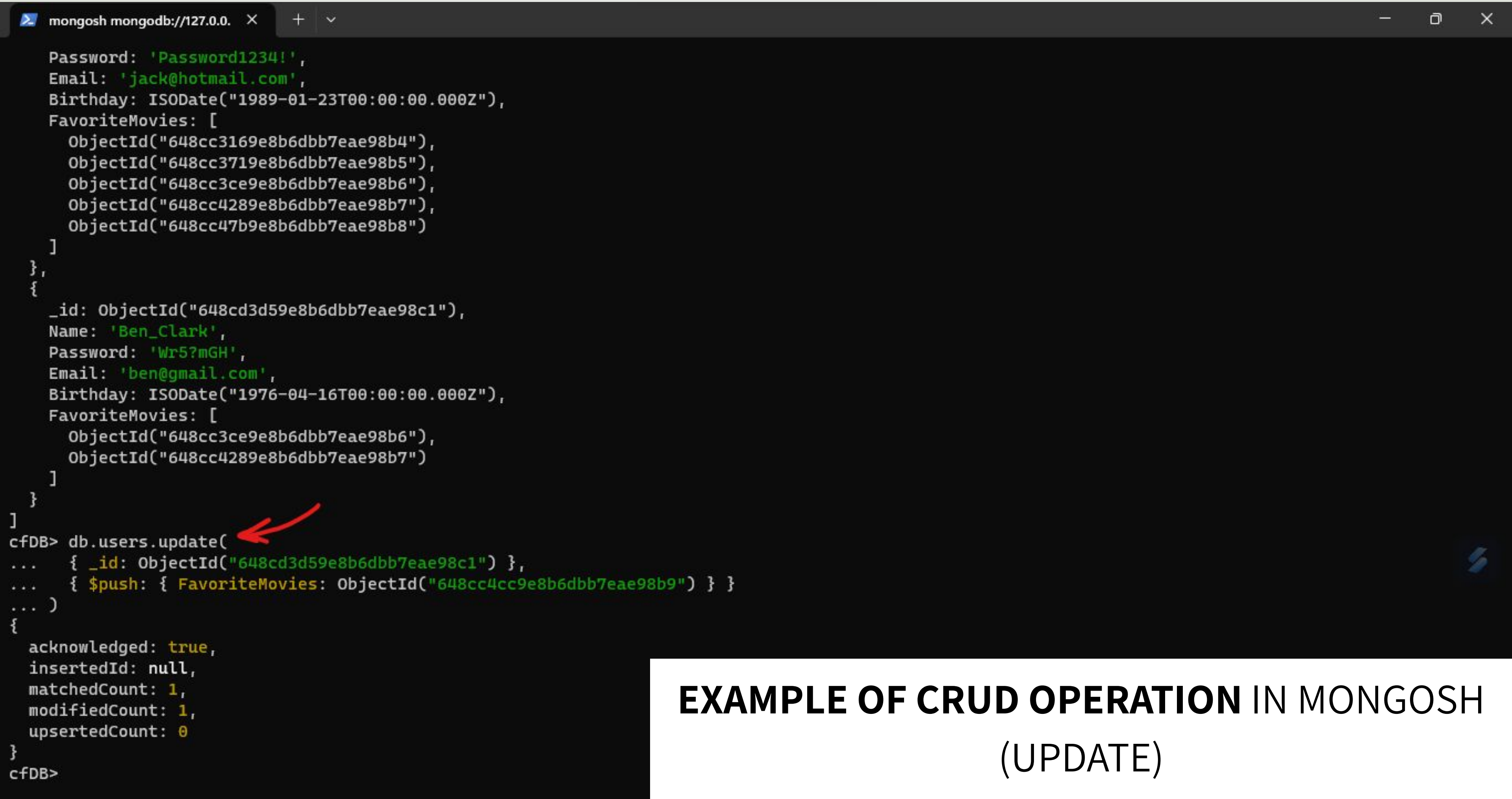
```
    Name: 'Thriller',
    Description: 'Thriller film, also known as suspense film or suspense thriller, is a broad film genre that involves excitement and suspense in th
e audience.'
  },
  Director: {
    Name: 'Steven Spielberg',
    Bio: 'Steven Spielberg is an American filmmaker and a major figure of the New Hollywood era.',
    Birth: '1946-12-18',
    Death: 'NA'
  },
  ImagePath: 'https://www.imdb.com/title/tt0073195/mediaviewer/rm1449540864/?ref_=tt_ov_i',
  Featured: false
}
cfDB> db.movies.find({ "Genre.Name": "Fantasy" })
[
  {
    _id: ObjectId("648cc47b9e8b6dbb7eae98b8"),
    Title: 'Princess Mononoke',
    Description: 'Princess Mononoke is a 1997 Japanese animated epic historical fantasy film.',
    Genre: {
      Name: 'Fantasy',
      Description: 'Fantasy films are films that belong to the fantasy genre with fantastic themes, usually magic, supernatural events, mythology, f
olklore, or exotic fantasy worlds.'
    },
    Director: {
      Name: 'Hayao Miyazaki',
      Bio: 'Hayao Miyazaki is a Japanese animator, filmmaker, and manga artist.',
      Birth: '1941-01-05',
      Death: 'NA'
    },
    ImagePath: 'https://www.imdb.com/title/tt0119698/mediaviewer/
    Featured: false
  }
]
cfDB>
```

## **EXAMPLE OF CRUD OPERATION** IN MONGOSH (READ)



```
... })
[
  {
    _id: ObjectId("648cc4289e8b6dbb7eae98b7"),
    Title: 'Ratatouille',
    Description: "Set in Paris, the plot follows a young rat Remy (Oswalt) who dreams of becoming a chef at Auguste Gusteau's restaurant and tries t
o achieve his goal by forming an unlikely alliance with the restaurant's garbage boy Alfredo Linguini.",
    Genre: {
      Name: 'Comedy',
      Description: 'Comedy is a genre of film in which the main emphasis is on humor. These films are designed to make the audience laugh through am
usement and most often work by exaggerating characteristics for humorous effect.'
    },
    Director: {
      Name: 'Brad Bird',
      Bio: 'Brad Bird is an American film director, animator, screenwriter, producer and voice actor.',
      Birth: '1957-09-24',
      Death: 'NA'
    },
    ImagePath: 'https://www.imdb.com/title/tt0382932/mediaviewer/rm937921792/?ref_=tt_ov_i',
    Featured: false
  }
]
cfDB> db.movies.updateOne(
...    { _id: ObjectId("648cc3ce9e8b6dbb7eae98b6") },
...    { $set: { Description: "Set during the Sierra Leone Civil War from 1991 to 2002, the film depicts a country torn apart by the struggle between
 government loyalists and insurgent forces. It also portrays many of the atrocities of that war." } }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
cfDB>
```

## **EXAMPLE OF CRUD OPERATION** IN MONGOSH (UPDATE)



```
    Password: 'Password1234!',
    Email: 'jack@hotmail.com',
    Birthday: ISODate("1989-01-23T00:00:00.000Z"),
    FavoriteMovies: [
      ObjectId("648cc3169e8b6dbb7eae98b4"),
      ObjectId("648cc3719e8b6dbb7eae98b5"),
      ObjectId("648cc3ce9e8b6dbb7eae98b6"),
      ObjectId("648cc4289e8b6dbb7eae98b7"),
      ObjectId("648cc47b9e8b6dbb7eae98b8")
    ]
  },
  {
    _id: ObjectId("648cd3d59e8b6dbb7eae98c1"),
    Name: 'Ben_Clark',
    Password: 'Wr5?mGH',
    Email: 'ben@gmail.com',
    Birthday: ISODate("1976-04-16T00:00:00.000Z"),
    FavoriteMovies: [
      ObjectId("648cc3ce9e8b6dbb7eae98b6"),
      ObjectId("648cc4289e8b6dbb7eae98b7")
    ]
  }
]
cfDB> db.users.update(
...   { _id: ObjectId("648cd3d59e8b6dbb7eae98c1") },
...   { $push: { FavoriteMovies: ObjectId("648cc4cc9e8b6dbb7eae98b9") } }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
cfDB>
```

**EXAMPLE OF CRUD OPERATION** IN MONGOSH (UPDATE)

## 08

**Creating** schemas and models to **enforce** data uniformity and data consistency in the non-relational database

Skills used

Research
Code writing
Debugging

## **WHY** WAS THIS STEP IMPORTANT

Creating models was important to ensure to keep the data as consistent as possible, not only to have a good general structure, but also because every user who will use myFlix web app will expect to receive, for example, the same data format for each movie they'll look into. This was essential for a good and professional user experience.

## **WHAT** WAS THE GOAL

Learning how to use Mongoose to maintain the flexibility of a non-relational database while maintaining consistency throughout my database.

Creating a specific file for my models (model.js), importing Mongoose package into it and creating the schemas for the movies and users collection with specific keys-values to dictate the format of the documents to be created within these two collections.

## **WHAT** WAS THE GOAL (SUITE)

Creating the models that use the schemas previously defined in the same model.js file (movies and users schemas) and export them. Import them into my index.js file to ensure my API endpoints can make use of them in order to query my MongoDB database according to the schemas defined (so to enforce attributes as I create, update, and delete certain documents in the database).

Connecting Mongoose into my REST API with mongoose.connect(   ) to allow it to perform CRUD operations on my MongoDB data (documents). Querying my Mongoose models using some common Mongoose querying functions (findOne, updateOne, updateMany, deleteOne, etc.) to ensure requests coming from the client and messages sent back are working and match how I wanted to store / alter / deliver data in MongoDB.

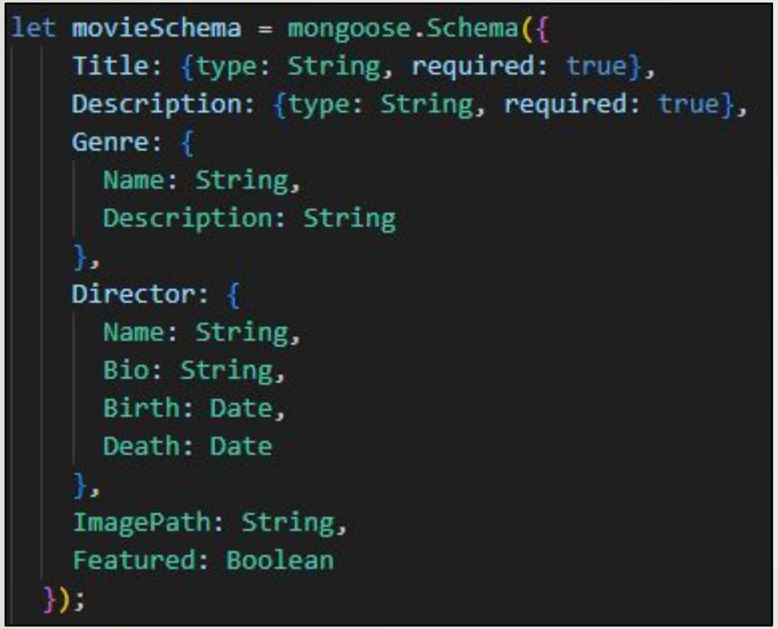
Testing back again each endpoint of my API using Postman to see if implemented changes (Mongoose models) are working as expected.

Updating my API documentation to integrate new and more accurate information based on the newly created schemas / models for my database.

## **CHALLENGES** OR SPECIAL POINTS OF CONSIDERATION

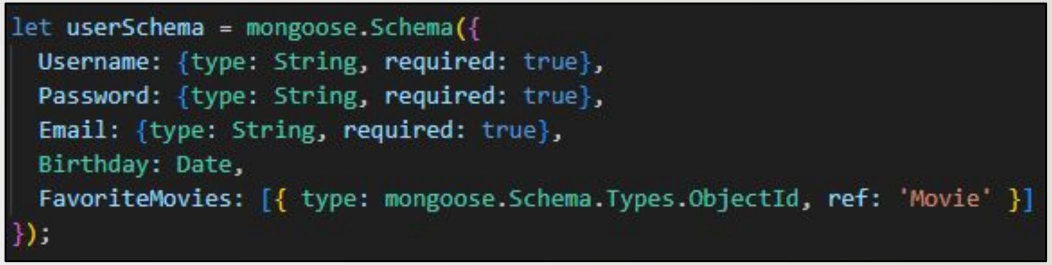
At the end of this step, I ran into some problems when trying to execute CRUD operations in Postman. I was always receiving a MongooseError message. I noticed that this was related to a Network / Connection issue, since I was not using the correct mongoose.connect( ) link to connect it to my local machine (issue related to the Mongoose version I had). I therefore looked online and found another mongoose.connect( ) link to allow Mongoose to connect to my local machine / database as expected, and it worked.

## **SCHEMA** FOR MOVIES COLLECTION



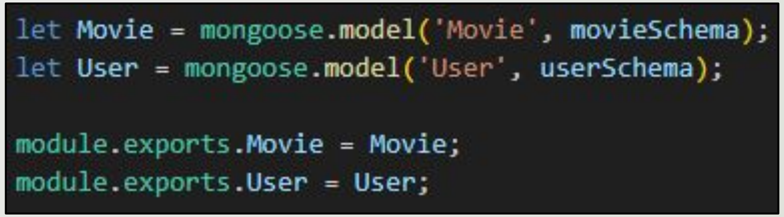
```
let movieSchema = mongoose.Schema({
    Title: {type: String, required: true},
    Description: {type: String, required: true},
    Genre: {
      Name: String,
      Description: String
    },
    Director: {
      Name: String,
      Bio: String,
      Birth: Date,
      Death: Date
    },
    ImagePath: String,
    Featured: Boolean
  });
```

## **SCHEMA** FOR USERS COLLECTION

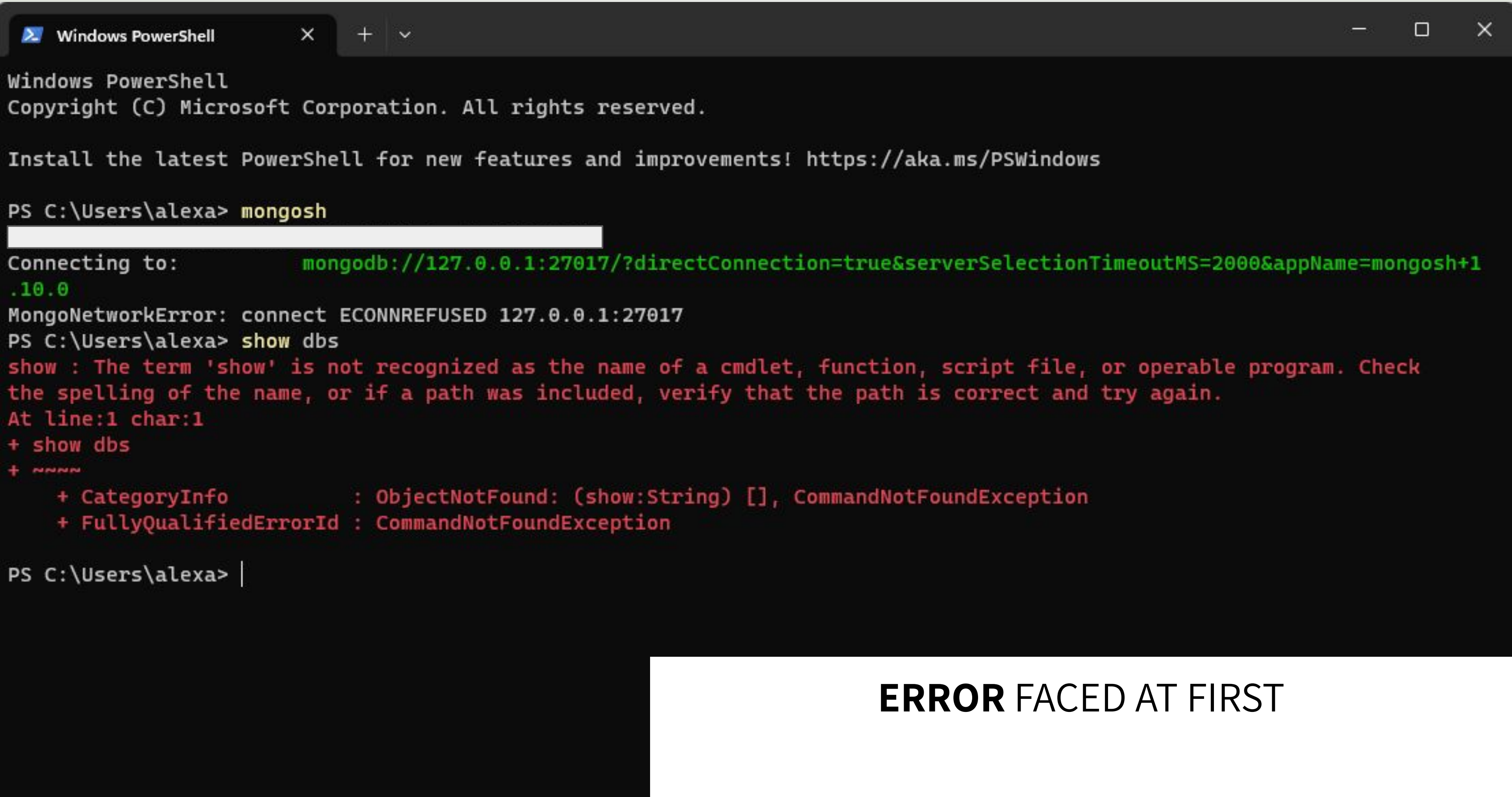


```
let userSchema = mongoose.Schema({
  Username: {type: String, required: true},
  Password: {type: String, required: true},
  Email: {type: String, required: true},
  Birthday: Date,
  FavoriteMovies: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Movie' }]
});
```

## **MODELS** CREATION AND EXPORT USING DEFINED SCHEMAS

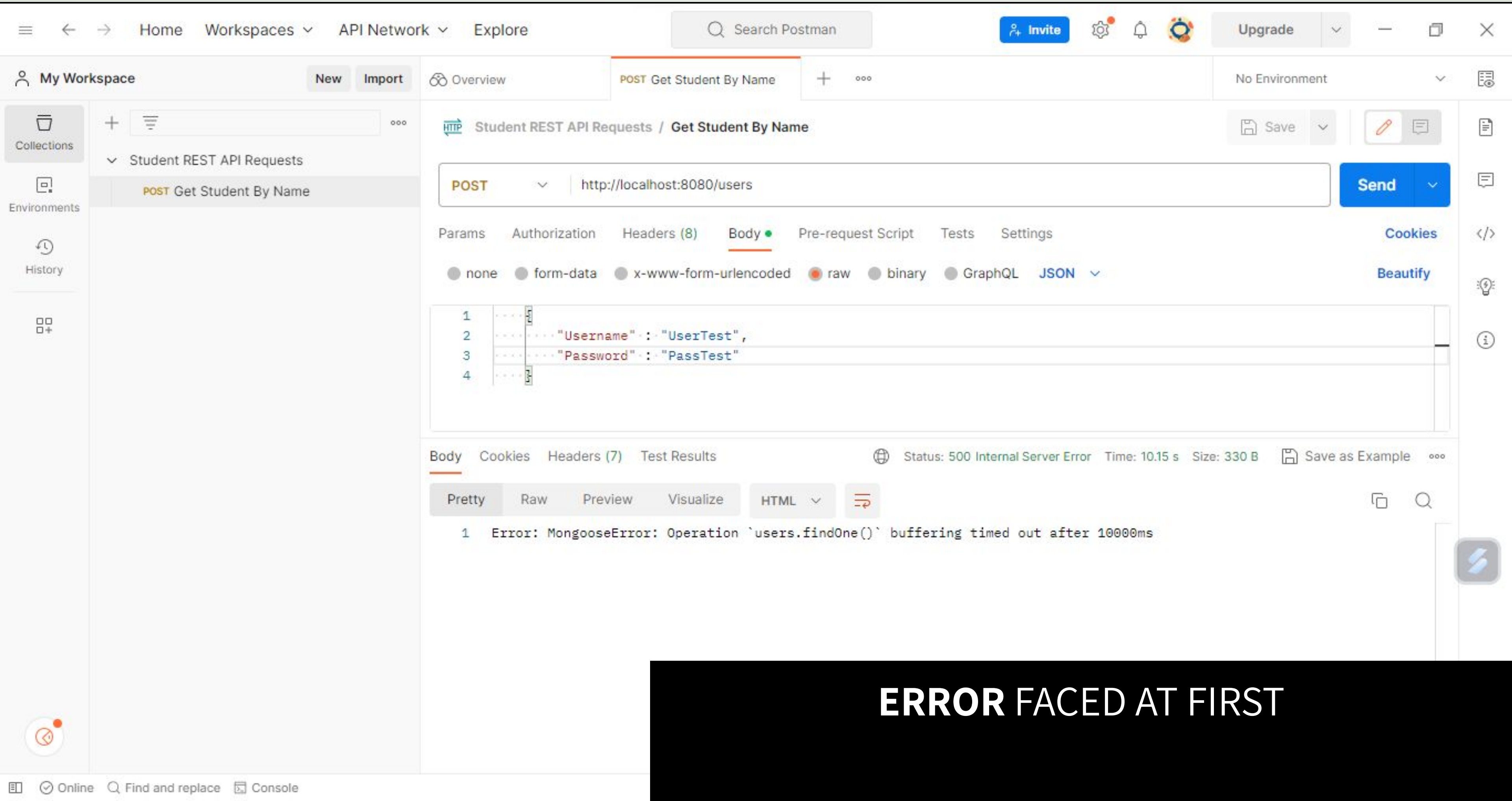


```
let Movie = mongoose.model('Movie', movieSchema);
let User = mongoose.model('User', userSchema);

module.exports.Movie = Movie;
module.exports.User = User;
```



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\alexa> mongosh
[redacted]
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1
.10.0
MongoNetworkError: connect ECONNREFUSED 127.0.0.1:27017
PS C:\Users\alexa> show dbs
show : The term 'show' is not recognized as the name of a cmdlet, function, script file, or operable program. Check
the spelling of the name, or if a path was included, verify that the path is correct and try again.
At line:1 char:1
+ show dbs
+ ~~~~
    + CategoryInfo          : ObjectNotFound: (show:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\alexa>
```

## **ERROR** FACED AT FIRST



Student REST API Requests / Get Student By Name

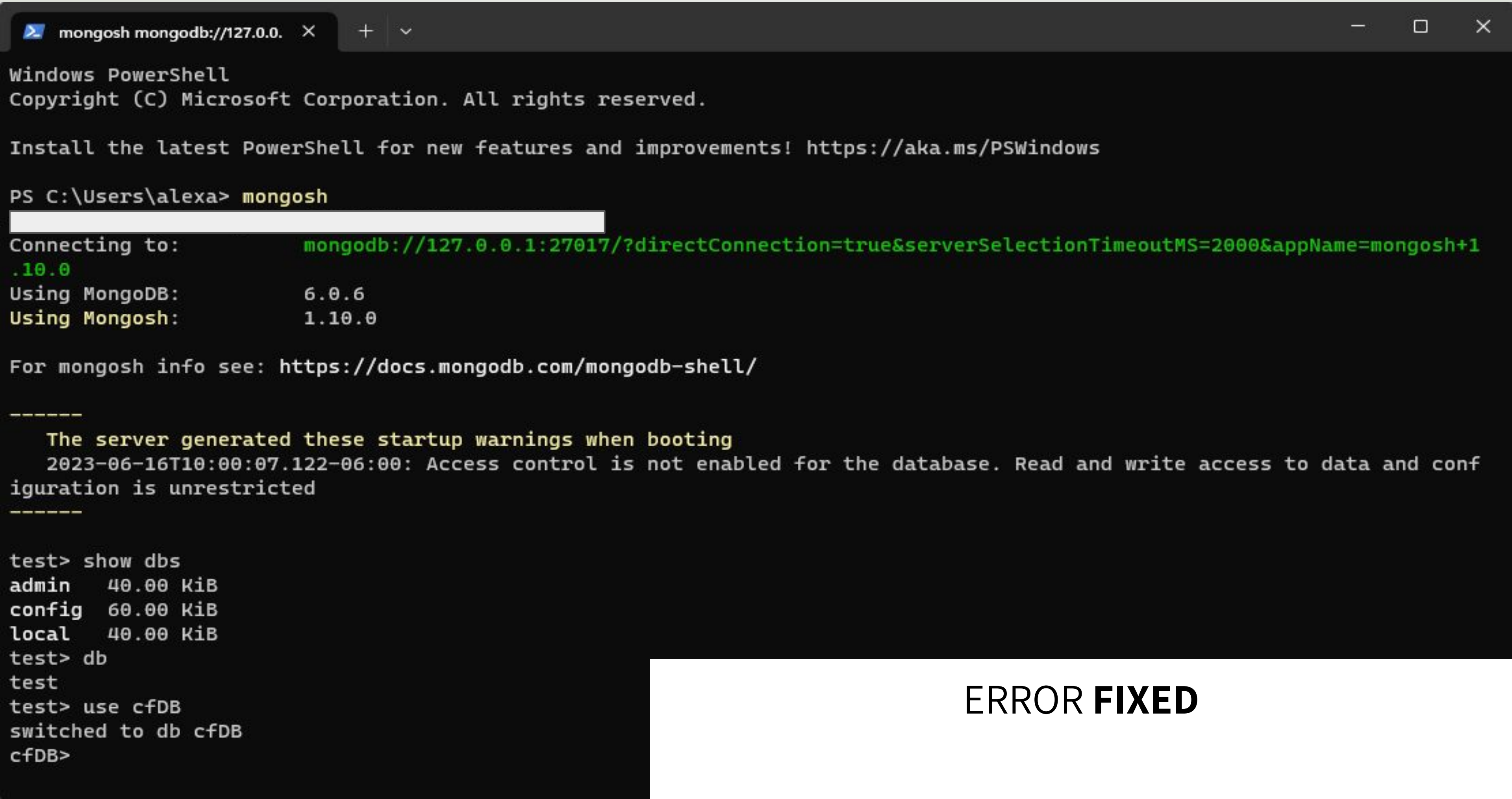
POST http://localhost:8080/users

```
{
    "Username" : "UserTest",
    "Password" : "PassTest"
}
```

Status: 500 Internal Server Error Time: 10.15 s Size: 330 B

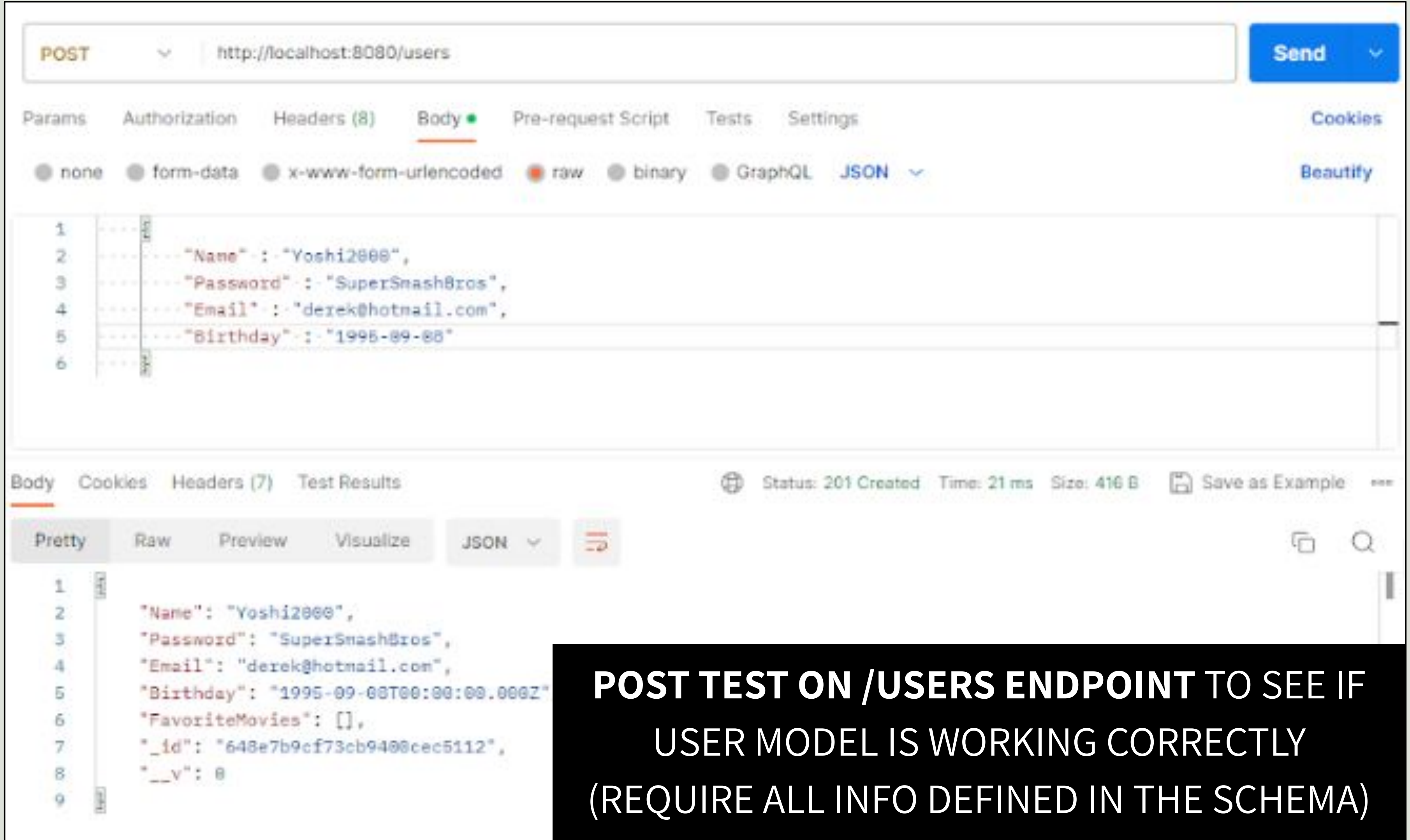
```
Error: MongooseError: Operation `users.findOne()` buffering timed out after 10000ms
```

## **ERROR** FACED AT FIRST



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\alexa> mongosh
[redacted]
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1
.10.0
Using MongoDB:          6.0.6
Using Mongosh:          1.10.0

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2023-06-16T10:00:07.122-06:00: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
------

test> show dbs
admin   40.00 KiB
config  60.00 KiB
local   40.00 KiB
test> db
test
test> use cfDB
switched to db cfDB
cfDB>
```

ERROR **FIXED**



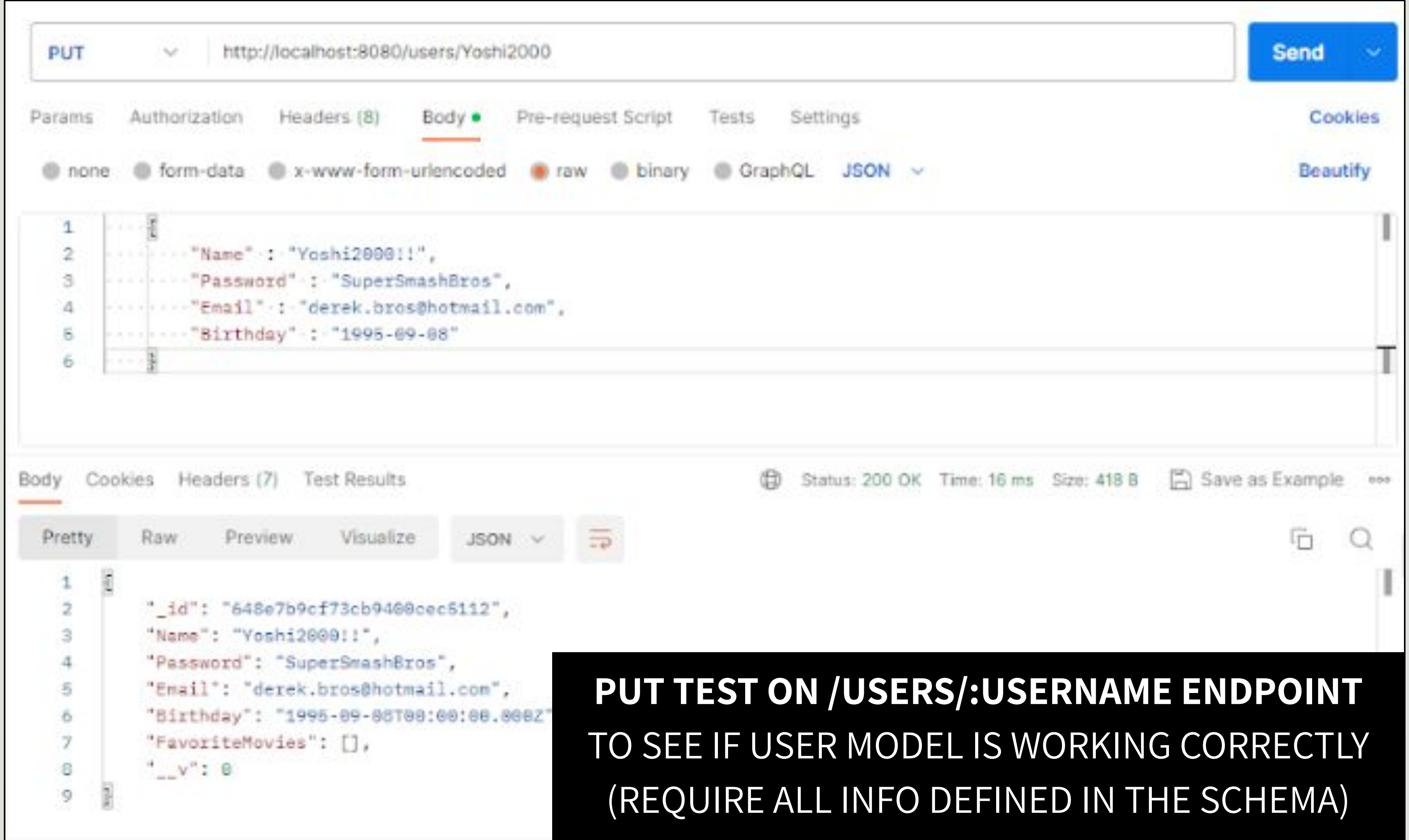
http://localhost:8080/users

```
{
    "Name" : "Yoshi2000",
    "Password" : "SuperSmashBros",
    "Email" : "derek@hotmail.com",
    "Birthday" : "1995-09-08"
}
```

Status: 201 Created  Time: 21 ms  Size: 416 B

```
{
    "Name": "Yoshi2000",
    "Password": "SuperSmashBros",
    "Email": "derek@hotmail.com",
    "Birthday": "1995-09-08T00:00:00.000Z",
    "FavoriteMovies": [],
    "_id": "648e7b9cf73cb9408cec5112",
    "__v": 0
}
```

**POST TEST ON /USERS ENDPOINT** TO SEE IF USER MODEL IS WORKING CORRECTLY (REQUIRE ALL INFO DEFINED IN THE SCHEMA)


PUT http://localhost:8080/users/Yoshi2000

```
{
    "Name" : "Yoshi2000!!",
    "Password" : "SuperSmashBros",
    "Email" : "derek.bros@hotmail.com",
    "Birthday" : "1995-09-08"
}
```

Status: 200 OK Time: 16 ms Size: 418 B

```
{
    "_id": "648e7b9cf73cb9400cec5112",
    "Name": "Yoshi2000!!",
    "Password": "SuperSmashBros",
    "Email": "derek.bros@hotmail.com",
    "Birthday": "1995-09-08T00:00:00.000Z"
    "FavoriteMovies": [],
    "__v": 0
}
```

**PUT TEST ON /USERS/:USERNAME ENDPOINT**

TO SEE IF USER MODEL IS WORKING CORRECTLY (REQUIRE ALL INFO DEFINED IN THE SCHEMA)

09

HTTP Authentication

API key-based authentication

session-based / cookie-based authentication

JWT token-based authentication

OAuth authentication

**Applying** principles of authentication to the REST API

Skills used

Research
Code writing
Debugging

## **WHY** WAS THIS STEP IMPORTANT

Users today expect the apps they use to be safe and secure, making authentication and authorization logic implementation an essential step in the development process. It was also important to keep the API safe from malicious actors and software.

Passport

## **WHAT** WAS THE GOAL

Learning and understanding the pros and cons of different types of authentication and authorization methods for applications: basic HTTP authentication, API key-based authentication, session-based (or cookie-based) authentication, JWT token-based authentication and OAuth.

Installing Passport library and basic HTTP authentication / JWT authentication packages in the project via the terminal.

## **WHAT** WAS THE GOAL (SUITE)

Creating a new file in my project folder (passport.js) and configuring two strategies using Passport middleware (strategies being Passport's block of codes to enable certain methods of authentication and authorization within an app).

- One for basic HTTP authentication to authenticate login requests (for users initial login requests into myFlix using username and password on the login page)
- One for JWT authentication to authenticate logged-in users requests based on their previously acquired JWT upon connection (for users requests to my API once logged in into myFlix)

Creating a new /login endpoint inside a new auth.js file to authenticate login requests using basic HTTP authentication and generate a JWT for future users requests / interactions with the API.

```
passport.use(new LocalStrategy(...));
passport.use(new JWTStrategy({...}));
```

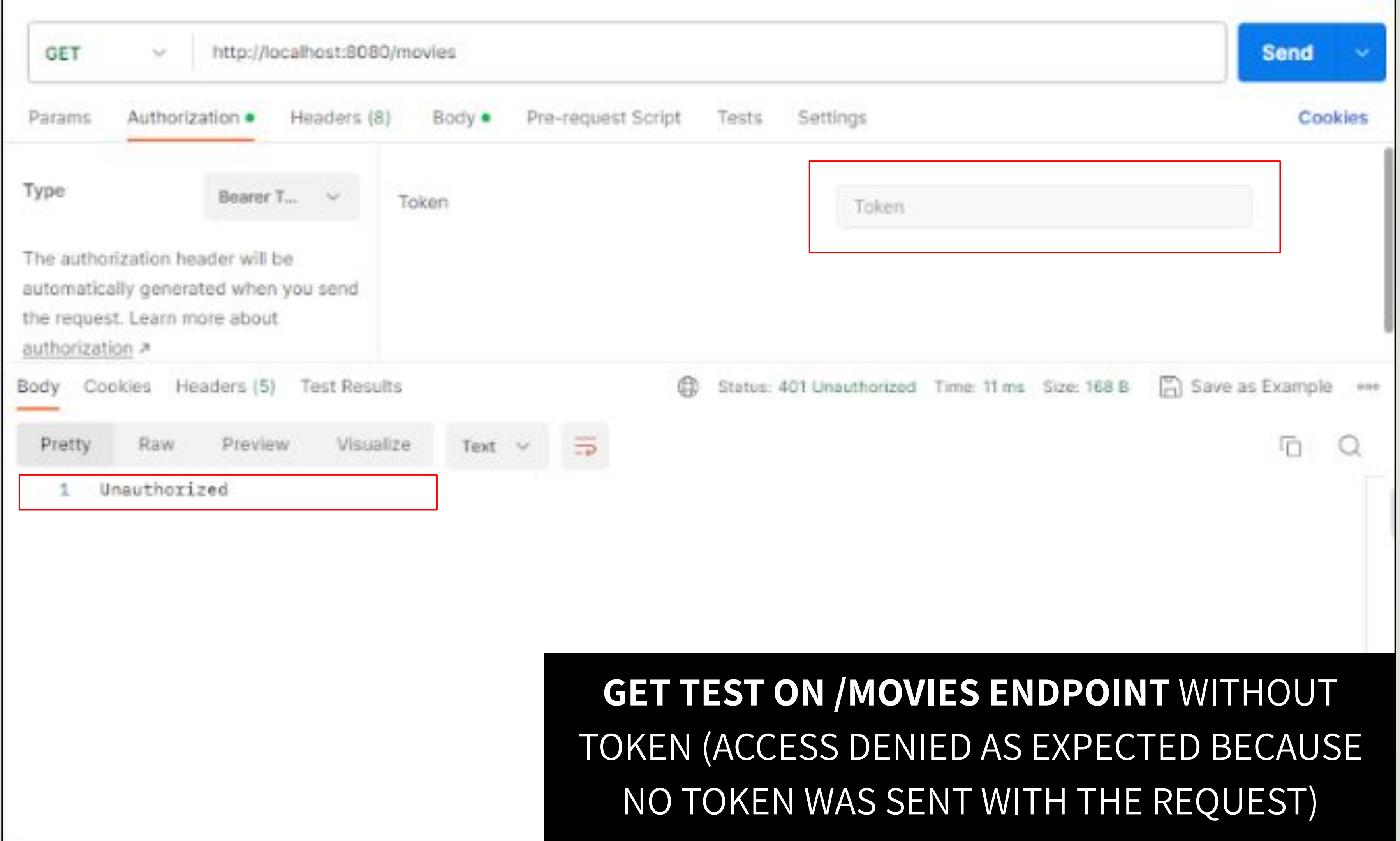
## **WHAT** WAS THE GOAL (SUITE)

Updating all my endpoints (except the ones for signing up and logging in) to integrate my JWT Passport strategy as middleware, so that only users with a JWT token can make requests to my API (for example, only users who've registered, been authenticated and send along automatically a token with their requests once in the app would be able to access the /movies endpoint and read the movie data sent back by the API).

This update / additional logic in every endpoint was meant for them to receive the token from the client-side and compare the details it contains with the details stored in the database to finally, if everything is matching, authorize the request to the endpoint and perform the specific CRUD operation.

Testing the new authentication and authorization methods implemented using Postman.

```
app.get('/movies', passport.authenticate('jwt', { session: false }), (req, res) => {...});
```


GET http://localhost:8080/movies

Send

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

Type: Bearer T...

The authorization header will be automatically generated when you send the request. Learn more about authorization ↗

Token: Token

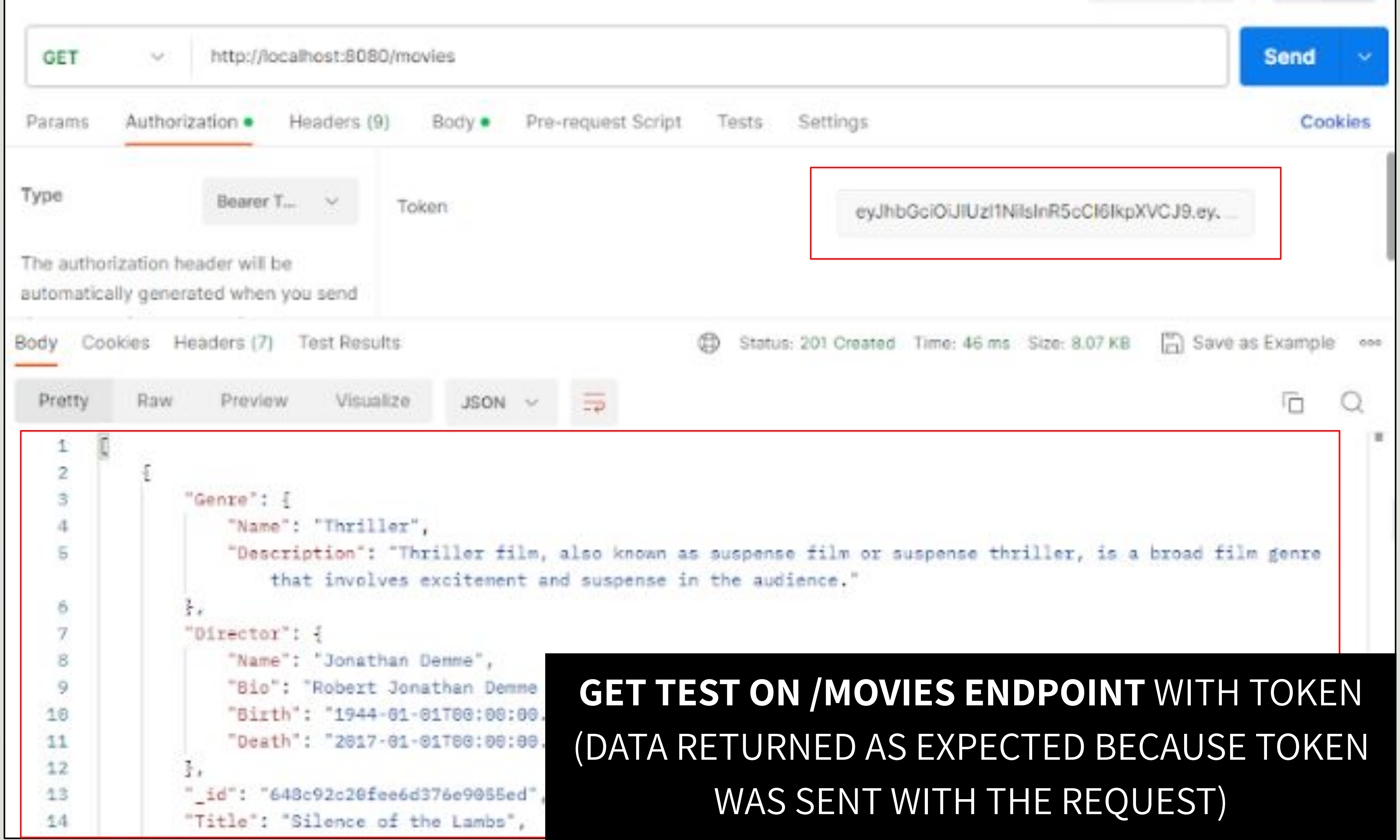
Body | Cookies | Headers (5) | Test Results

Status: 401 Unauthorized Time: 11 ms Size: 168 B Save as Example

Pretty | Raw | Preview | Visualize | Text

```
Unauthorized
```

**GET TEST ON /MOVIES ENDPOINT** WITHOUT TOKEN (ACCESS DENIED AS EXPECTED BECAUSE NO TOKEN WAS SENT WITH THE REQUEST)



http://localhost:8080/movies

Type: Bearer T...

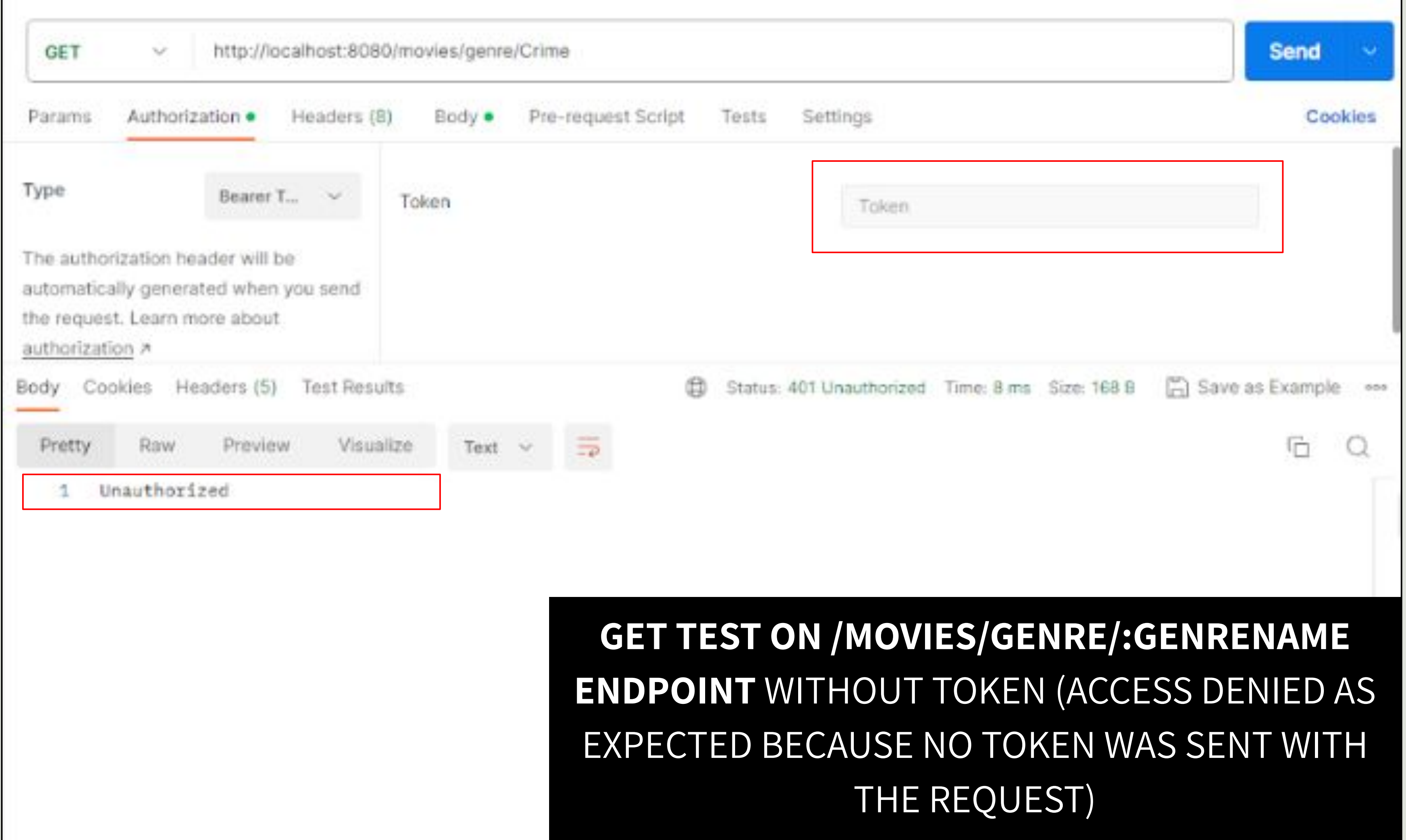
The authorization header will be automatically generated when you send

Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey...

Status: 201 Created Time: 46 ms Size: 8.07 KB

```
[
    {
        "Genre": {
            "Name": "Thriller",
            "Description": "Thriller film, also known as suspense film or suspense thriller, is a broad film genre
                that involves excitement and suspense in the audience."
        },
        "Director": {
            "Name": "Jonathan Demme",
            "Bio": "Robert Jonathan Demme
            "Birth": "1944-01-01T00:00:00.
            "Death": "2017-01-01T00:00:00.
        },
        "_id": "648c92c20fee6d376e9065ed",
        "Title": "Silence of the Lambs",
```

**GET TEST ON /MOVIES ENDPOINT** WITH TOKEN (DATA RETURNED AS EXPECTED BECAUSE TOKEN WAS SENT WITH THE REQUEST)



```
GET http://localhost:8080/movies/genre/Crime
```

Type: Bearer T...

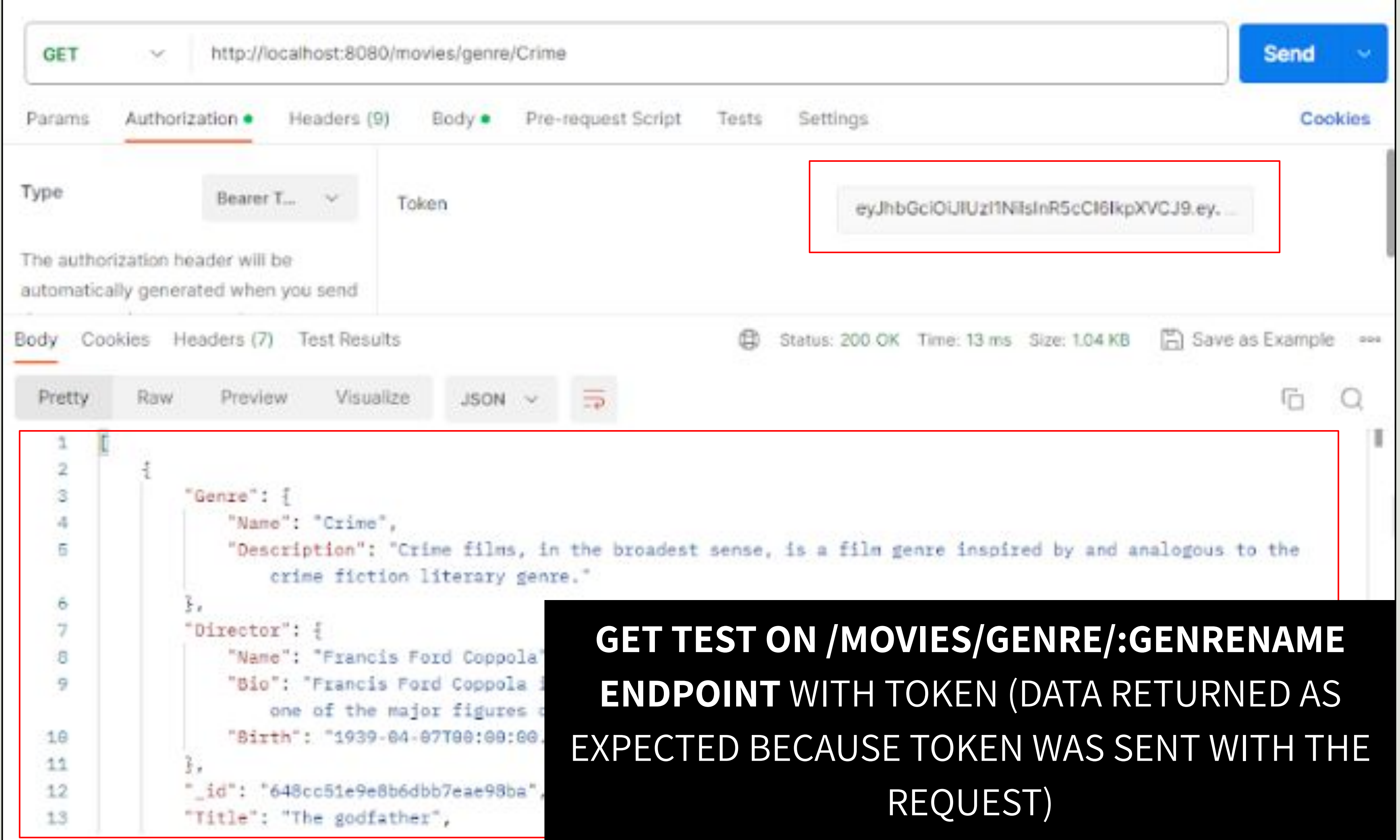
Token: Token

The authorization header will be automatically generated when you send the request. Learn more about authorization ↗

Status: 401 Unauthorized Time: 8 ms Size: 168 B

```
Unauthorized
```

**GET TEST ON /MOVIES/GENRE/:GENRENAME ENDPOINT** WITHOUT TOKEN (ACCESS DENIED AS EXPECTED BECAUSE NO TOKEN WAS SENT WITH THE REQUEST)



GET http://localhost:8080/movies/genre/Crime

Type: Bearer T...

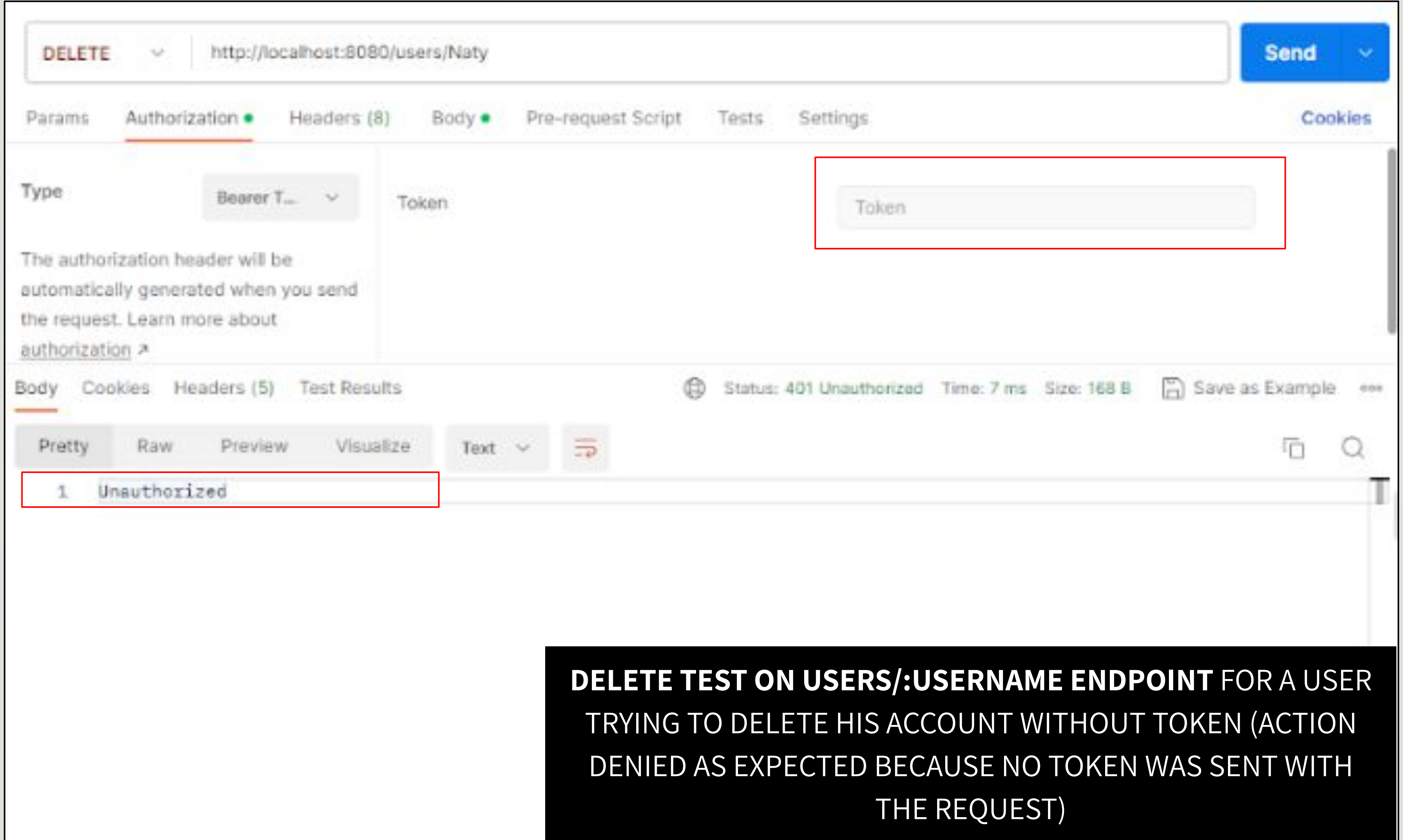
Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey...

The authorization header will be automatically generated when you send

Status: 200 OK Time: 13 ms Size: 1.04 KB

```
[
    {
        "Genre": {
            "Name": "Crime",
            "Description": "Crime films, in the broadest sense, is a film genre inspired by and analogous to the
                crime fiction literary genre."
        },
        "Director": {
            "Name": "Francis Ford Coppola
            "Bio": "Francis Ford Coppola i
                one of the major figures o
            "Birth": "1939-04-07T00:00:00.
        },
        "_id": "648cc51e9e8b6dbb7eae98ba",
        "Title": "The godfather",
```

**GET TEST ON /MOVIES/GENRE/:GENRENAME ENDPOINT** WITH TOKEN (DATA RETURNED AS EXPECTED BECAUSE TOKEN WAS SENT WITH THE REQUEST)



DELETE http://localhost:8080/users/Naty

Type: Bearer T...

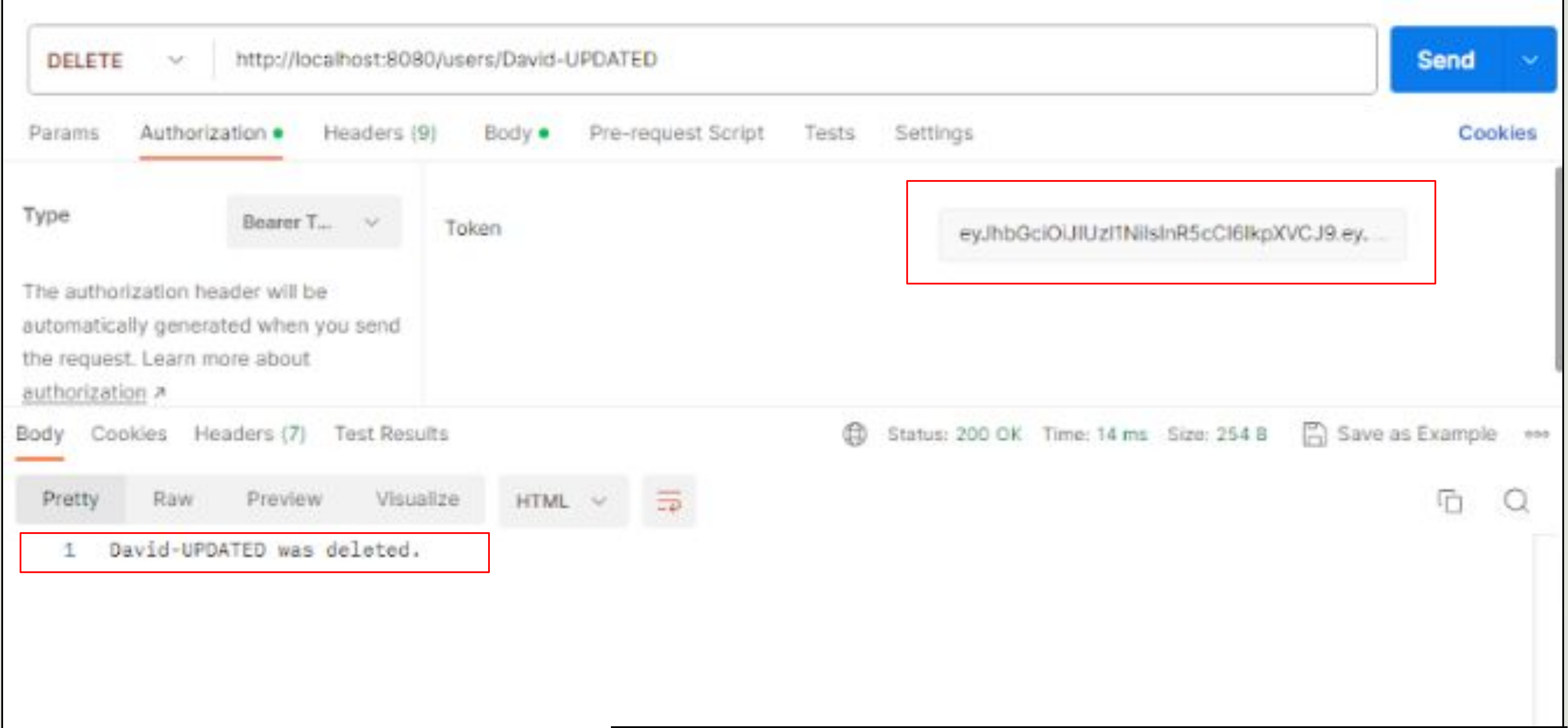
Token: Token

The authorization header will be automatically generated when you send the request. Learn more about authorization ↗

Status: 401 Unauthorized Time: 7 ms Size: 168 B

```
Unauthorized
```

**DELETE TEST ON USERS/:USERNAME ENDPOINT** FOR A USER TRYING TO DELETE HIS ACCOUNT WITHOUT TOKEN (ACTION DENIED AS EXPECTED BECAUSE NO TOKEN WAS SENT WITH THE REQUEST)



DELETE http://localhost:8080/users/David-UPDATED

Type: Bearer T...

Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey...

The authorization header will be automatically generated when you send the request. Learn more about authorization ↗

Status: 200 OK Time: 14 ms Size: 254 B

```
David-UPDATED was deleted.
```

**DELETE TEST ON USERS/:USERNAME ENDPOINT** FOR A USER TRYING TO DELETE HIS ACCOUNT WITH TOKEN (ACTION COMPLETED AS EXPECTED BECAUSE TOKEN WAS SENT WITH THE REQUEST)

10

## **Implementing** security mechanisms and **deploying** the API and database online

Skills used

Research
Problem-solving
Code writing
Debugging

## **WHY** WAS THIS STEP IMPORTANT

Ensuring data and web security considerations have been incorporated into the web app was important, since web developers and whole teams behind digital product development have security and ethical responsibilities (such as privacy laws and data protection measures).

- `Same-Origin Policy`
- `Cross-Origin Resource Sharing (CORS)`
- `HTTPS`
- `Secure Sockets Layer (SSL)`
- `Password hashing`
- `Content-Security-Policy (CSP)`
- `User input validation`
- `Escaping data`

- `Cross-Site Scripting Attacks (XSS)`
- `Cross-Site Request Forgery (CSRF)`
- `SQL Injection`

## **WHAT** WAS THE GOAL

Learn about privacy and ethical laws, especially coming from the GDPR.

Learning about security mechanisms for the web:

- Same-Origin Policy
- Cross-Origin Resource Sharing (CORS)
- HTTPS
- Secure Sockets Layer (SSL)
- Password hashing
- Content-Security-Policy (CSP)
- User input validation
- Escaping data

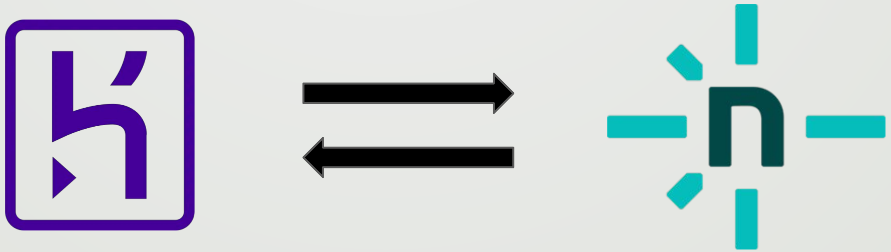
Learning about different forms of malicious attacks:

- Cross-Site Scripting Attacks (XSS)
- Cross-Site Request Forgery (CSRF)
- SQL Injection

## **WHAT** WAS THE GOAL (SUITE)

Installing and implementing restricting domain access to keep my app as safe as possible from malicious entities using Cross-Origin Resource Sharing (CORS) module. This is a useful module as it extends HTTP requests sent to my API by giving them a new header that include their origin domain, thus allowing the server to then identify where the requests are coming from (which domain) and allow or disallow the request accordingly.

For this to work, I  added  (later during my app frontend development) the links of my publicly hosted web app (React and Angular) into my CORS domain permissions (which then listed the origin domains for my web app frontends as authorized domains). This allowed anyone going on my web app via Netlify (my hosting platform for my React frontend) or gh-pages (my hosting platform for my Angular frontend) to access it, while other domain origins trying to access it, potentially being malicious, could not (error message returned).



## **WHAT** WAS THE GOAL (SUITE)

Installing and implementing the Bcrypt module to hash myFlix users' password in the database upon account creation.

- Then comparing hashed passwords received from users login requests to the hashed password stored into the database to ensure a more secure login authentication process.

```
Password selected by user during account creation: password1234!
Password hashed and stored in the database: $4n$93$ynFKZA5247XqHjZbTwRmAqk7K28xGp41DR7CvQsH
```

## **WHAT** WAS THE GOAL (SUITE)

Installing Express-validator and implementing server-side input validation logics on any endpoints that expect data in the request body to ensure only accepted characters and formats submitted by users make their way into the database, thus protecting it from potential harmful inputs / actions. Such validation includes for example:

- Only alphanumeric characters accepted for username (.isAlphanumeric Express-validator method)
- Requiring specific elements / part structure for emails (.isEmail Express-validator method)

Deploying / hosting my API on the Platform as a Service (PaaS) Heroku and deploying / hosting my database online into cloud-based database hosting platform MongoDB Atlas.

Connecting my database to my API on Heroku, ensuring that the entire backend of myFlix is online, connected and ready to be used.

Final testing of all possible requests in Postman using my API's URLs from Heroku.

## **CHALLENGES** OR SPECIAL POINTS OF CONSIDERATION

The biggest challenge has been to understand all the different security measures that can be used to protect an application from malicious attacks, and to understand what each of these solutions brings as advantages. At the end, I understood the logic of these measures, how they work and implemented some of them into my app, but I still have a lot to know in cybersecurity. This is a particularly dense subject, for which I would like to continue learning.

express-validator

11

## **Finalizing  code revision** and refactoring

Skills used

Critical thinking
Detailed overview

## **WHY** WAS THIS STEP IMPORTANT

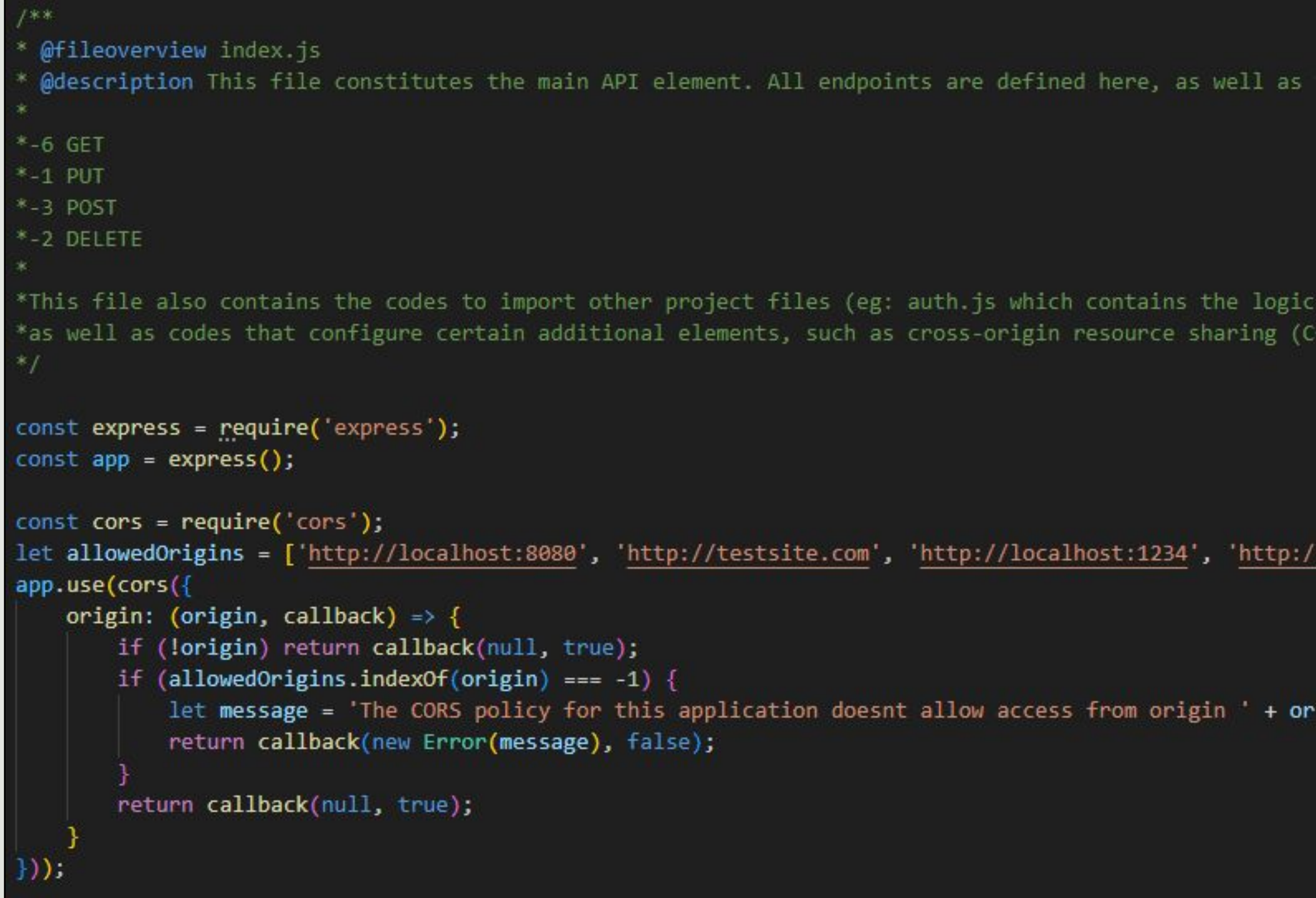
Ensuring that the codes are optimized to facilitate possible appropriation by other developers in the future is useful and could possibly save time. It can also facilitate any future adjustments to the codes.

## **WHAT** WAS THE GOAL

Reviewing each code to make sure everything was optimized as much as possible in order to facilitate future modifications, additions or adjustments in the future.

Adding comments and clarification points in the code where important for the benefit and better understanding of anyone else who may work on this project later.

## **CODE COMMENTS** TO FACILITATE FUTURE UPDATES AND WORK



```
/**
* @fileoverview index.js
* @description This file constitutes the main API element. All endpoints are defined here, as well as
*
*-6 GET
*-1 PUT
*-3 POST
*-2 DELETE
*
*This file also contains the codes to import other project files (eg: auth.js which contains the logic
*as well as codes that configure certain additional elements, such as cross-origin resource sharing (C
*/

const express = require('express');
const app = express();

const cors = require('cors');
let allowedOrigins = ['http://localhost:8080', 'http://testsite.com', 'http://localhost:1234', 'http:/
app.use(cors({
    origin: (origin, callback) => {
        if (!origin) return callback(null, true);
        if (allowedOrigins.indexOf(origin) === -1) {
            let message = 'The CORS policy for this application doesnt allow access from origin ' + or
            return callback(new Error(message), false);
        }
        return callback(null, true);
    }
}));
```

## **CHALLENGES** OR SPECIAL POINTS OF CONSIDERATION

I positioned myself from the point of view of future colleagues who could work on my project. How can I make my project and my codes as clear as possible to promote its easy appropriation? I reviewed each file to bring improvements in certain places and add comments where I thought it could be useful.

## **DECISIONS** MADE

This step was done on my own initiative and was not required in the project requirements. I made decisions regarding the improvement of certain parts of my codes, and the addition of comments where necessary in order to set my mind to work in a collaborative environment already.

README.
md

## **Completing** the README document

Skills used

Communication
Content writing

## **WHY** WAS THIS STEP IMPORTANT

Ensure myFlix backend logic  is well documented and easily accessible by anyone interested.

## **WHAT** WAS THE GOAL

Updating and completing the README file located in the myFlix Github repository. The goal was to ensure that all relevant information regarding myFlix is accessible under these four categories:

- Project description
- Technical aspects
- List of endpoints and related information
- App dependencies

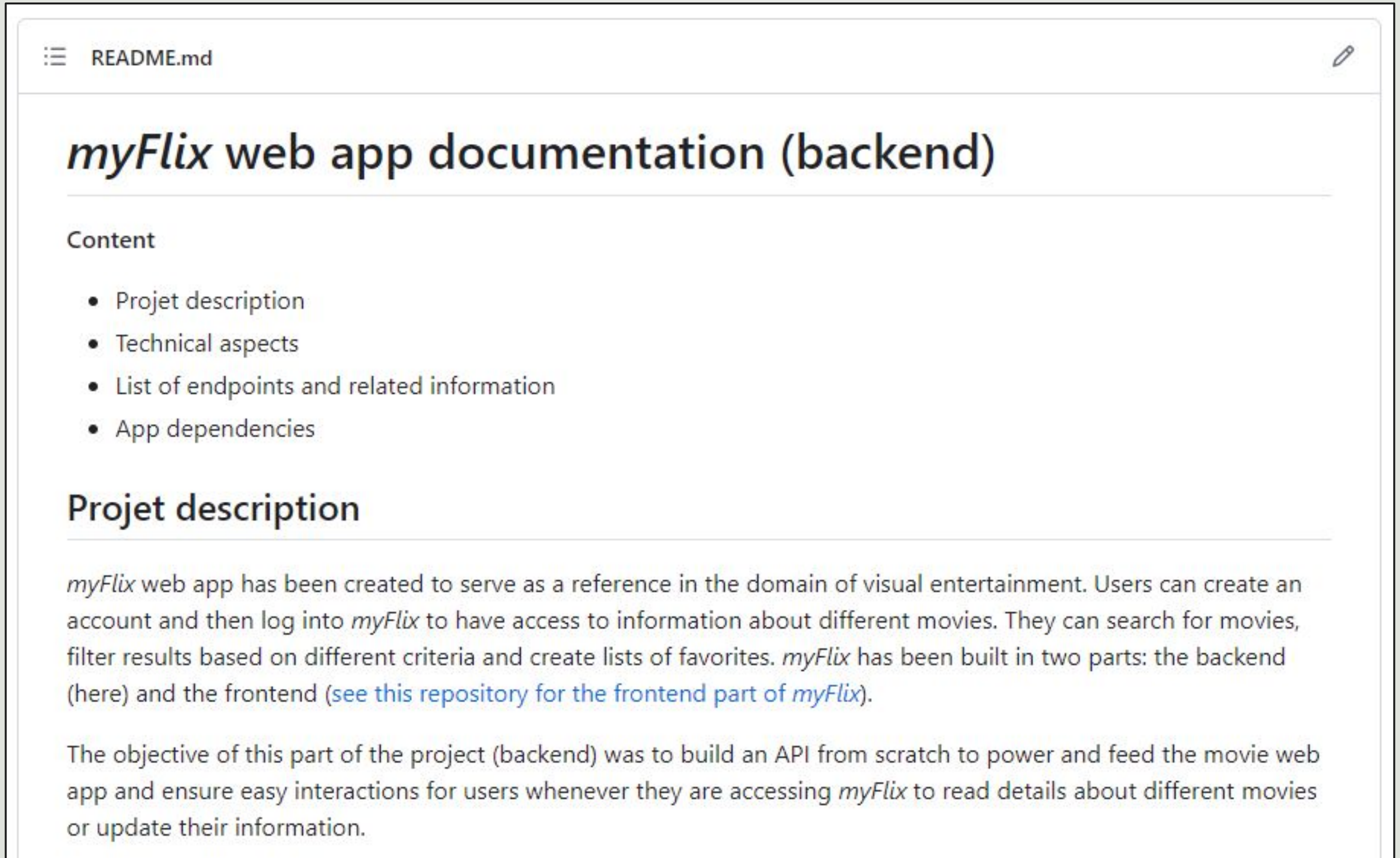
## **CHALLENGES** OR SPECIAL POINTS OF CONSIDERATION

Finding the right balance between giving the right level of information, while remaining as synthetic as possible. To help me, I made a first draft, which I then modified several times. I get inspired by other READMEs I've consulted for similar projects and for which I found that the information presented was relevant.

## **DECISIONS** MADE

I wrote the README documentation from A to Z, in terms of content, presentation and structure.

## **README SAMPLE** - FULL VERSION ON GITHUB



README.md

# *myFlix* web app documentation (backend)

**Content**

- Projet description
- Technical aspects
- List of endpoints and related information
- App dependencies

## Projet description

*myFlix* web app has been created to serve as a reference in the domain of visual entertainment. Users can create an account and then log into *myFlix* to have access to information about different movies. They can search for movies, filter results based on different criteria and create lists of favorites. *myFlix* has been built in two parts: the backend (here) and the frontend (see this repository for the frontend part of *myFlix*).

The objective of this part of the project (backend) was to build an API from scratch to power and feed the movie web app and ensure easy interactions for users whenever they are accessing *myFlix* to read details about different movies or update their information.