Royal University of Bhutan

College of Science and Technology
Rinchending: Bhutan

DBS101
Database Systems Fundamentals
SS(2024)

Practical{9}   Report

Submitted By;

Student Name : Tashi Penjor

Enrollment No.: 02230306

Programme : BESWE

Date : 30/04/2024

Table of content

Topic : Guided session

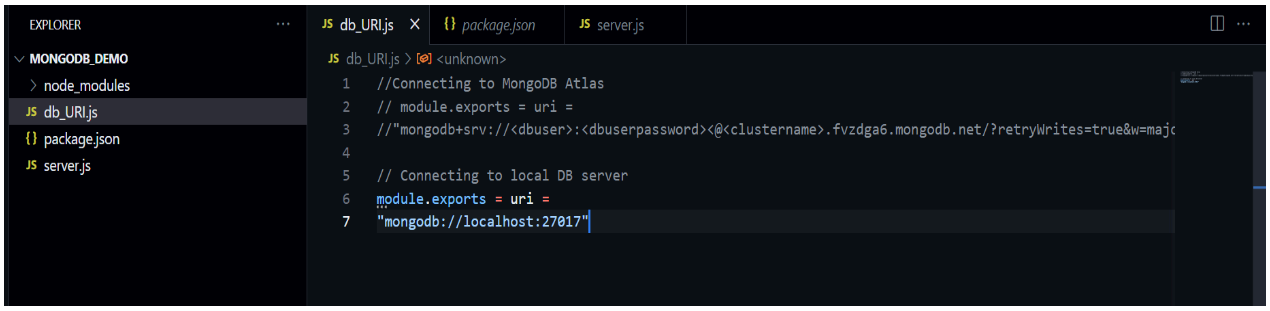
The guided session is about connecting the database using the npm.

Software that allows for communication between an application and a DBMS.

In MongoDB the driver works with the built in Node js BSON package to interact with the MongoDB server.

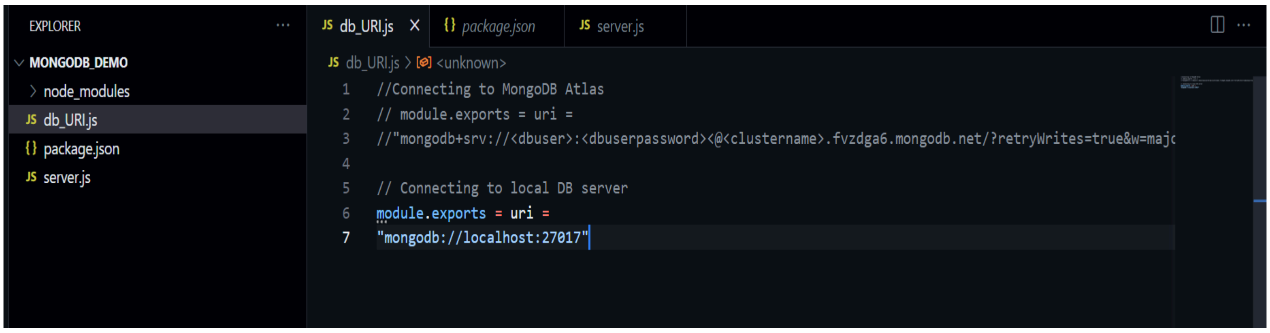
Task 1 : In this task I have created a new folder(MongoDB_Demo) to do the following task and store the database.

Task 2 : In the task it is all about downloading the dependencies to execute the task. After executing the npm init and npm install mongodb the package.json and the node module was downloaded in the folder.



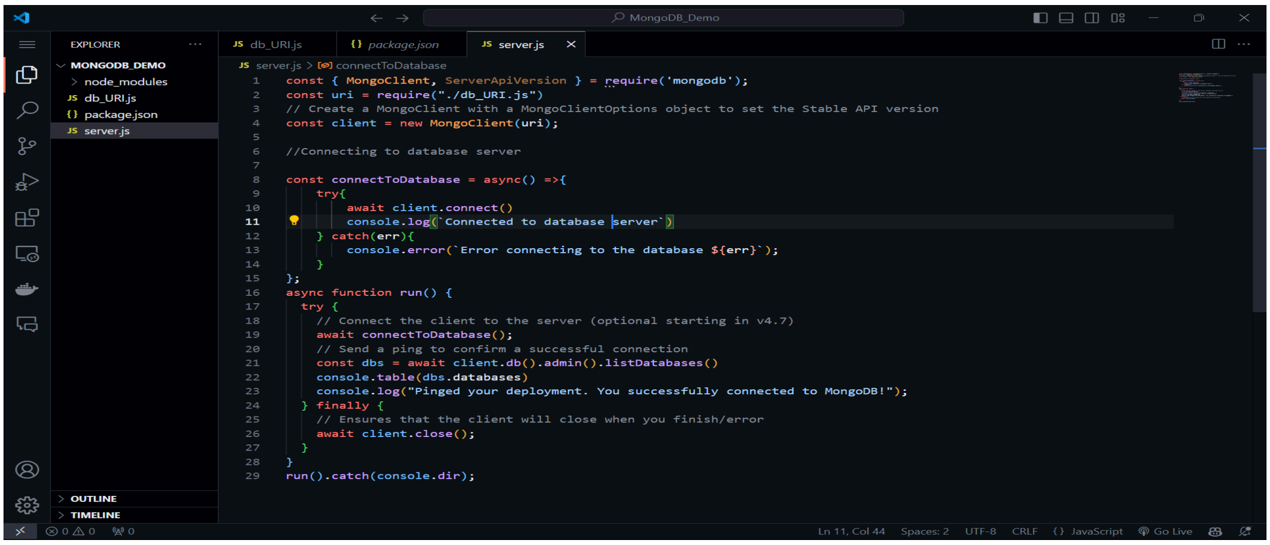
```
//Connecting to MongoDB Atlas
// module.exports = uri =
//"mongodb+srv://<dbuser>:<dbuserpassword>@<clustername>.fvzdga6.mongodb.net/?retryWrites=true&w=majo

// Connecting to local DB server
module.exports = uri =
"mongodb://localhost:27017"
```

Task 3 : In this task I have to create a db_URL.js and add mongoDB connection string. Connecting the string is crucial for connecting the mongoDB server to the application.



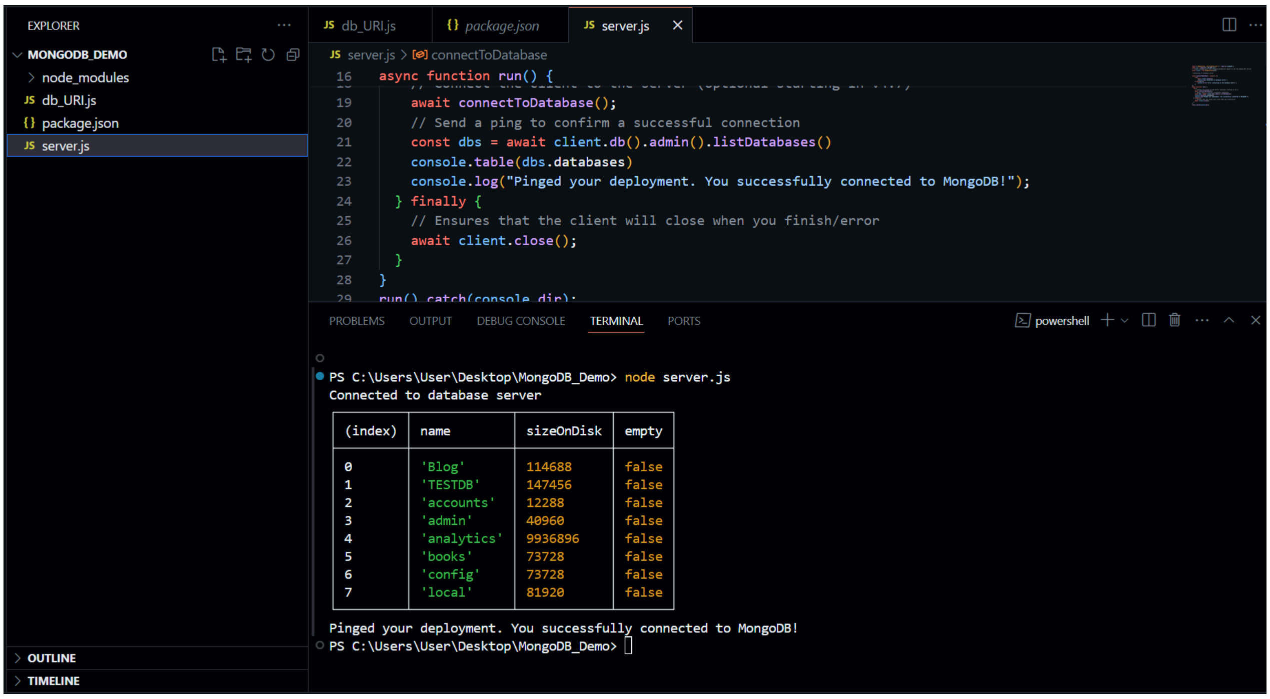
```
//Connecting to MongoDB Atlas
// module.exports = uri =
//"mongodb+srv://<dbuser>:<dbuserpassword>@<clustername>.fvzdga6.mongodb.net/?retryWrites=true&w=majo

// Connecting to local DB server
module.exports = uri =
"mongodb://localhost:27017"
```

Task 4 : In this task I have to create the server.js file to establish a database connection locally to the mongodb server.



```javascript
const { MongoClient, ServerApiVersion } = require('mongodb');
const uri = require("./db_URI.js")
// Create a MongoClient with a MongoClientOptions object to set the Stable API version
const client = new MongoClient(uri);

//Connecting to database server

const connectToDatabase = async() =>{
    try{
        await client.connect()
        console.log(`Connected to database server`)
    } catch(err){
        console.error(`Error connecting to the database ${err}`);
    }
};
async function run() {
  try {
    // Connect the client to the server (optional starting in v4.7)
    await connectToDatabase();
    // Send a ping to confirm a successful connection
    const dbs = await client.db().admin().listDatabases()
    console.table(dbs.databases)
    console.log("Pinged your deployment. You successfully connected to MongoDB!");
  } finally {
    // Ensures that the client will close when you finish/error
    await client.close();
  }
}
run().catch(console.dir);
```

Task 5 : In this task I have to run the server setup in the above steps and I got the result of all the collections in the mongodb server.



```
PS C:\Users\User\Desktop\MongoDB_Demo> node server.js
Connected to database server
┌─────────┬─────────────┬────────────┬───────┐
│ (index) │ name        │ sizeOnDisk │ empty │
├─────────┼─────────────┼────────────┼───────┤
│ 0       │ 'Blog'      │ 114688     │ false │
│ 1       │ 'TESTDB'    │ 147456     │ false │
│ 2       │ 'accounts'  │ 12288      │ false │
│ 3       │ 'admin'     │ 40960      │ false │
│ 4       │ 'analytics' │ 9936896    │ false │
│ 5       │ 'books'     │ 73728      │ false │
│ 6       │ 'config'    │ 73728      │ false │
│ 7       │ 'local'     │ 81920      │ false │
└─────────┴─────────────┴────────────┴───────┘
Pinged your deployment. You successfully connected to MongoDB!
PS C:\Users\User\Desktop\MongoDB_Demo>
```

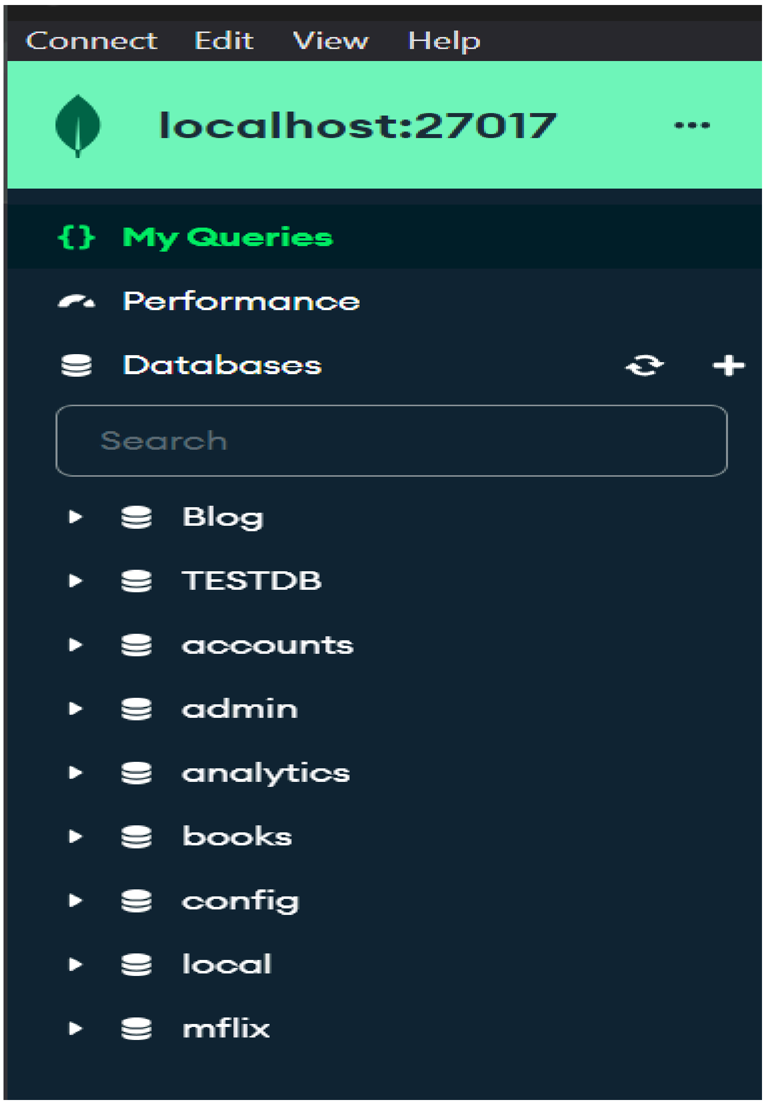
Conclusion :

The guided session provides a comprehensive overview of the connection to a mongodb database using the npm package in a Node.js environment. This process involves several key steps that are essential for setting up and managing a mongodb database connection within a Node.js application. Moreover it also demonstrates the process of setting up a mongodb database connection in a Node.js application using npm packages.

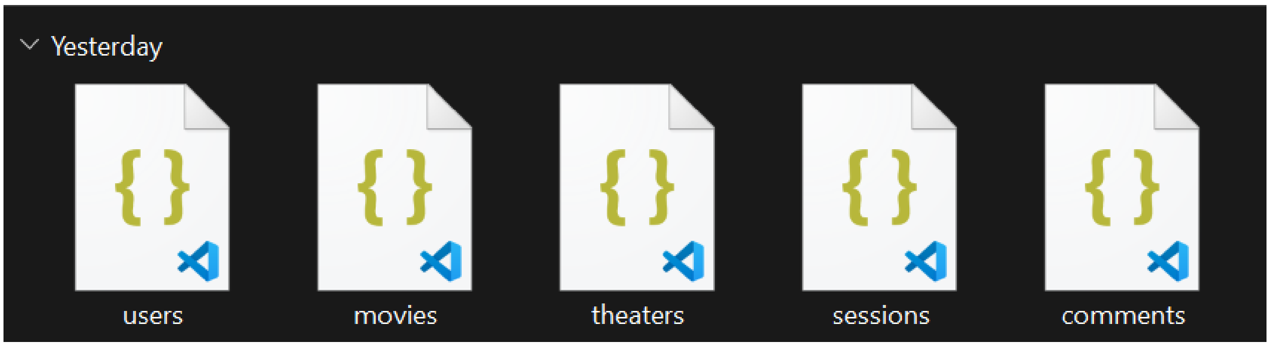
Topics : Exercise

Loading a sample dataset by performing the following operations

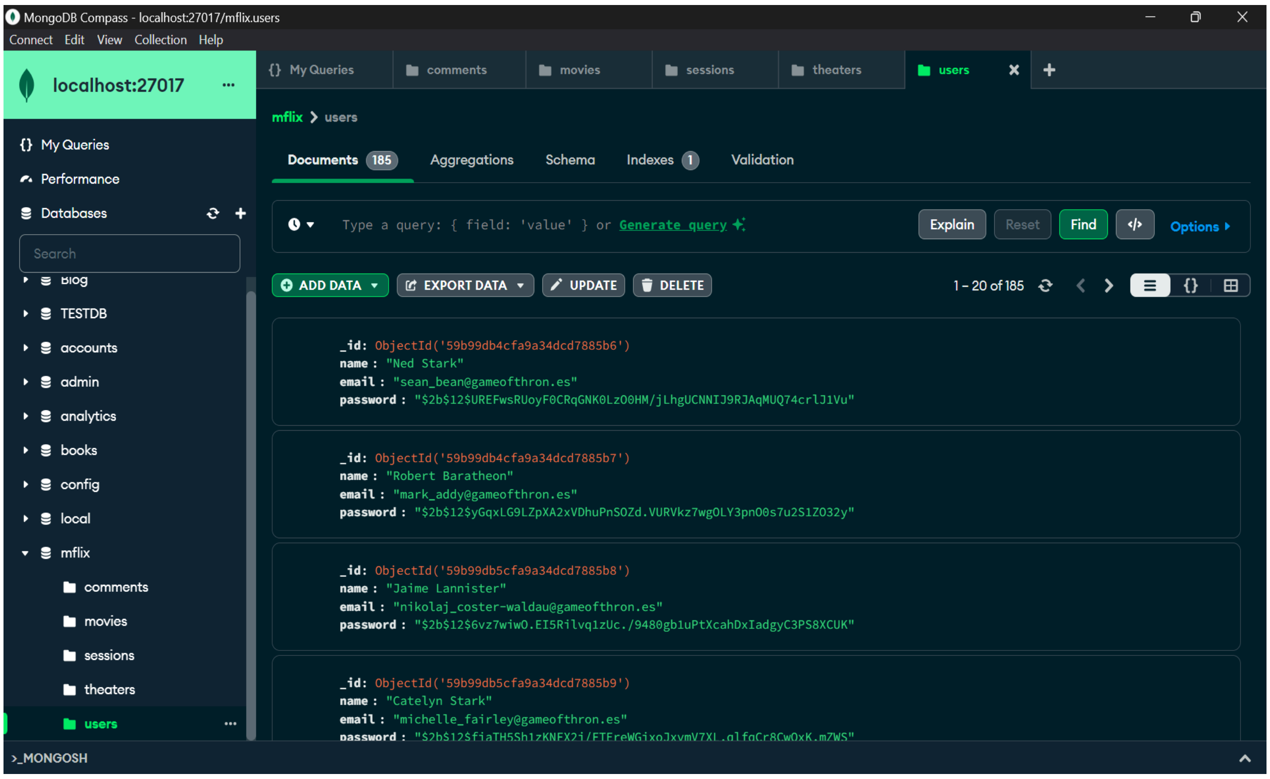
Task 1 : In this task I have created a database(mflix) to perform the following task.



Task 2 : In this task I downloaded the sample mflix database from the given github link.



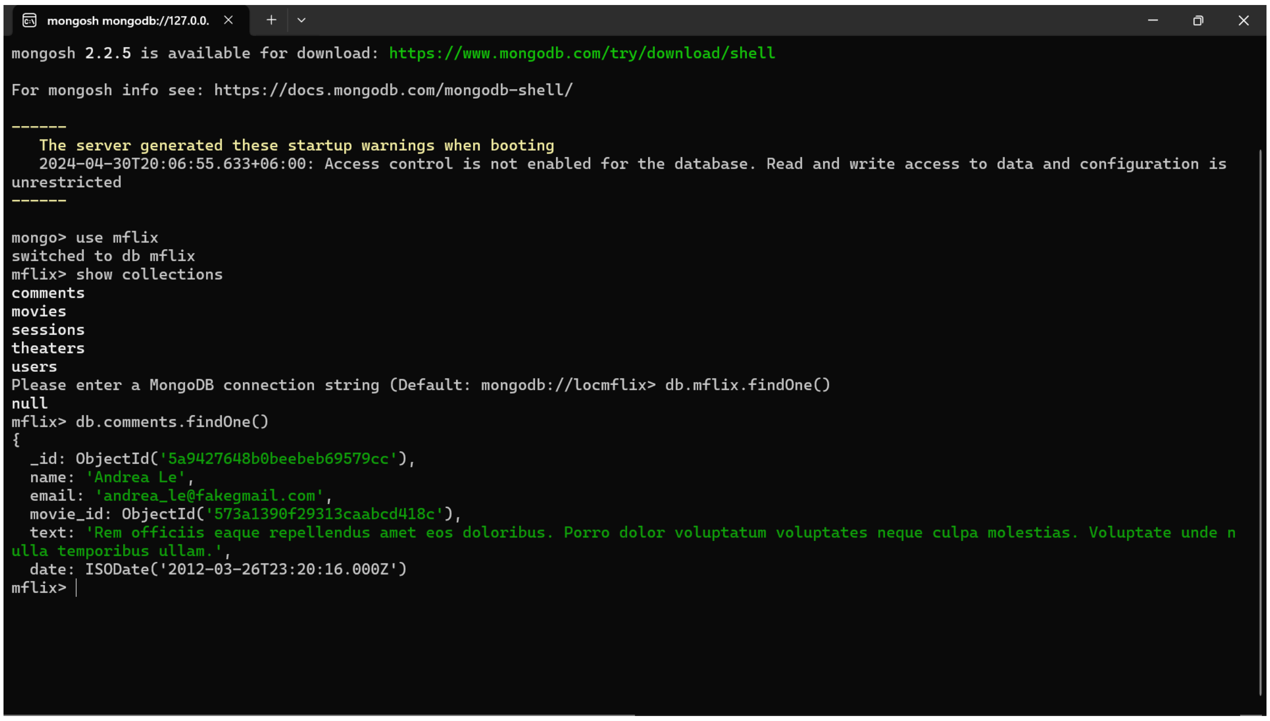
Task 3 : In this task I have created a collection of the sample I downloaded from the github under the mflix database.



```
_id: ObjectId('59b99db4cfa9a34dcd7885b6')
name : "Ned Stark"
email : "sean_bean@gameofthron.es"
password : "$2b$12$UREFwsRUoyF0CRqGNK0LzO0HM/jLhgUCNNIJ9RJAqMUQ74crlJ1Vu"

_id: ObjectId('59b99db4cfa9a34dcd7885b7')
name : "Robert Baratheon"
email : "mark_addy@gameofthron.es"
password : "$2b$12$yGqxLG9LZpXA2xVDhuPnSOZd.VURVkz7wgOLY3pn00s7u2S1Z032y"

_id: ObjectId('59b99db5cfa9a34dcd7885b8')
name : "Jaime Lannister"
email : "nikolaj_coster-waldau@gameofthron.es"
password : "$2b$12$6vz7wiwO.EI5Rilvq1zUc./9480gb1uPtXcahDxIadgyC3PS8XCUK"

_id: ObjectId('59b99db5cfa9a34dcd7885b9')
name : "Catelyn Stark"
email : "michelle_fairley@gameofthron.es"
password : "$2b$12$fiaTH5Sh1zKNFX2i/FTFreWGixoJxvmV7XL.qlfqCr8CwOxK.mZWS"
```

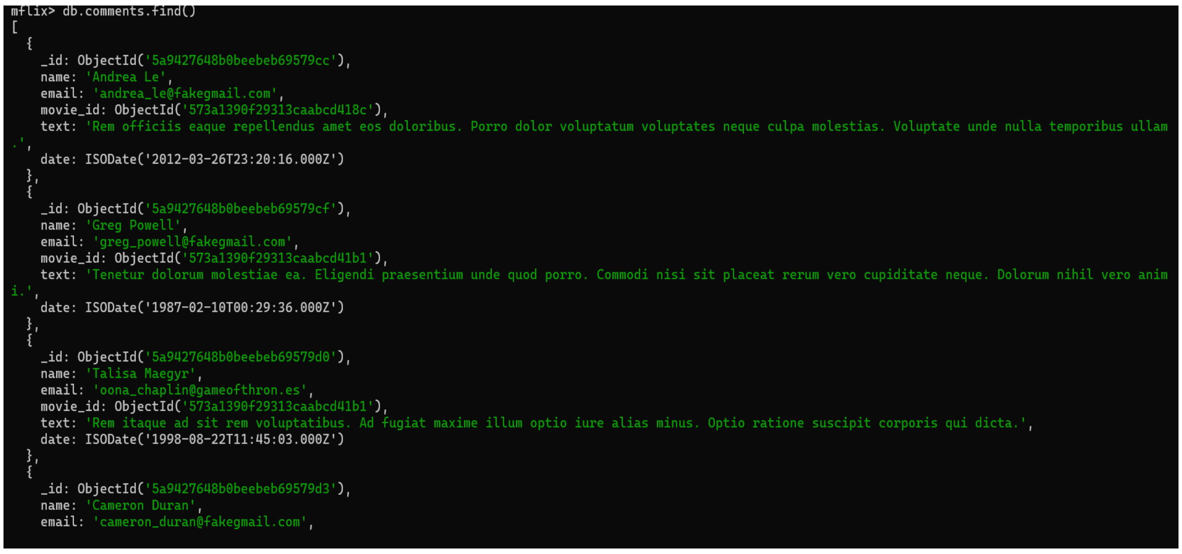
Task 4 : In this task I have done some operations

1) **findOne()** - displaying only one comment from the comments sample database.

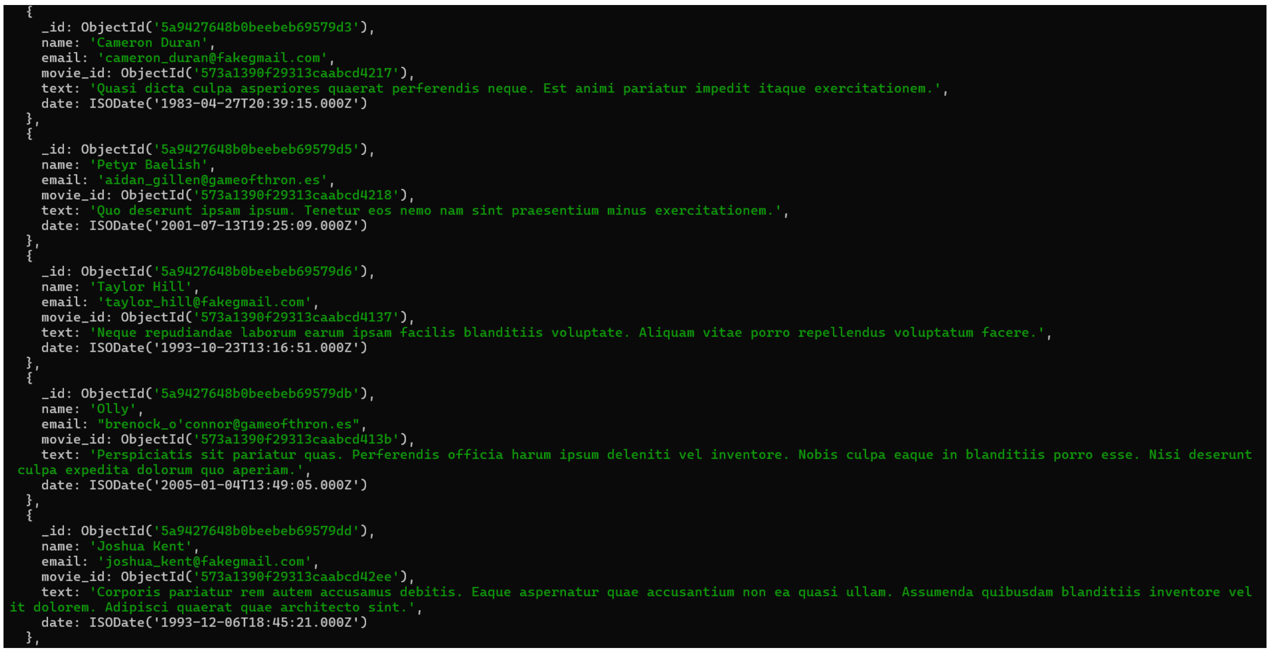


```
mongosh 2.2.5 is available for download: https://www.mongodb.com/try/download/shell

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

------
   The server generated these startup warnings when booting
   2024-04-30T20:06:55.633+06:00: Access control is not enabled for the database. Read and write access to data and configuration is
unrestricted
------

mongo> use mflix
switched to db mflix
mflix> show collections
comments
movies
sessions
theaters
users
Please enter a MongoDB connection string (Default: mongodb://locmflix> db.mflix.findOne()
null
mflix> db.comments.findOne()
{
  _id: ObjectId('5a9427648b0beebeb69579cc'),
  name: 'Andrea Le',
  email: 'andrea_le@fakegmail.com',
  movie_id: ObjectId('573a1390f29313caabcd418c'),
  text: 'Rem officiis eaque repellendus amet eos doloribus. Porro dolor voluptatum voluptates neque culpa molestias. Voluptate unde n
ulla temporibus ullam.',
  date: ISODate('2012-03-26T23:20:16.000Z')
mflix> 
```

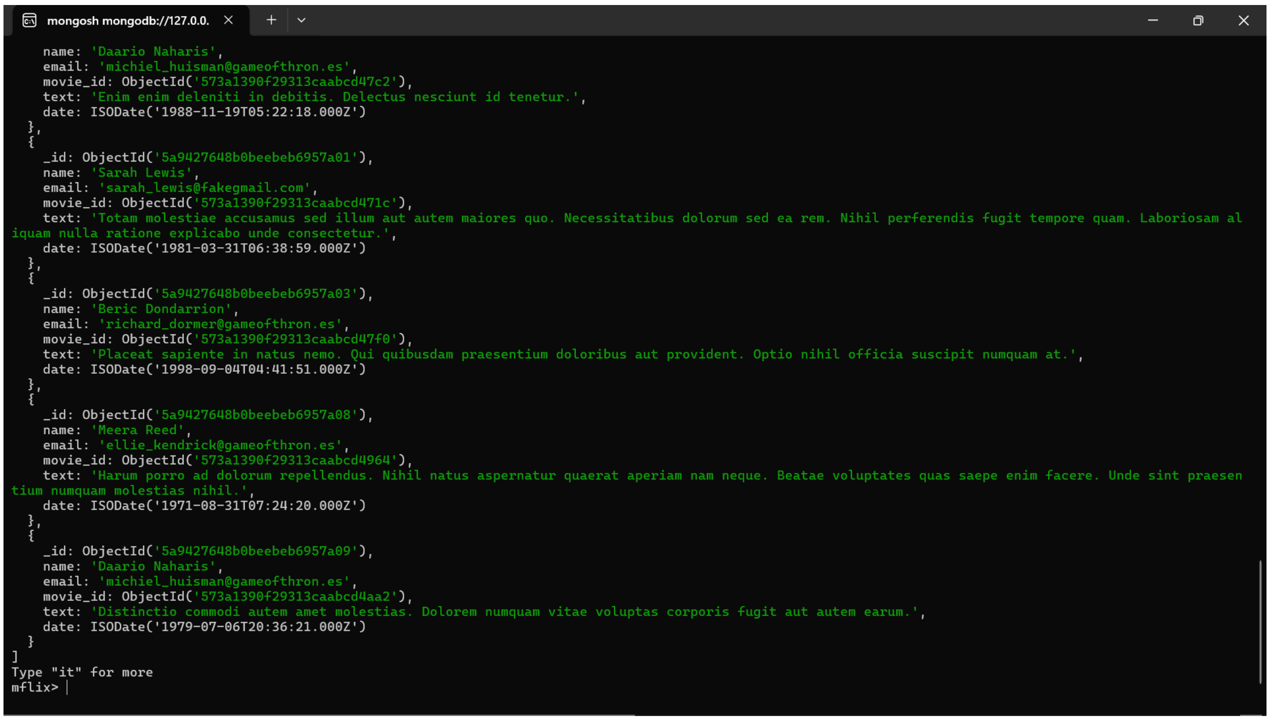
2) **find()** - displaying all the comments from the comments sample database.



```
mflix> db.comments.find()
[
  {
    _id: ObjectId('5a9427648b0beebeb69579cc'),
    name: 'Andrea Le',
    email: 'andrea_le@fakegmail.com',
    movie_id: ObjectId('573a1390f29313caabcd418c'),
    text: 'Rem officiis eaque repellendus amet eos doloribus. Porro dolor voluptatum voluptates neque culpa molestias. Voluptate unde nulla temporibus ullam
.',
    date: ISODate('2012-03-26T23:20:16.000Z')
  },
  {
    _id: ObjectId('5a9427648b0beebeb69579cf'),
    name: 'Greg Powell',
    email: 'greg_powell@fakegmail.com',
    movie_id: ObjectId('573a1390f29313caabcd41b1'),
    text: 'Tenetur dolorum molestiae ea. Eligendi praesentium unde quod porro. Commodi nisi sit placeat rerum vero cupiditate neque. Dolorum nihil vero anim
i.',
    date: ISODate('1987-02-10T00:29:36.000Z')
  },
  {
    _id: ObjectId('5a9427648b0beebeb69579d0'),
    name: 'Talisa Maegyr',
    email: 'oona_chaplin@gameofthron.es',
    movie_id: ObjectId('573a1390f29313caabcd41b1'),
    text: 'Rem itaque ad sit rem voluptatibus. Ad fugiat maxime illum optio iure alias minus. Optio ratione suscipit corporis qui dicta.',
    date: ISODate('1998-08-22T11:45:03.000Z')
  },
  {
    _id: ObjectId('5a9427648b0beebeb69579d3'),
    name: 'Cameron Duran',
    email: 'cameron_duran@fakegmail.com',
```



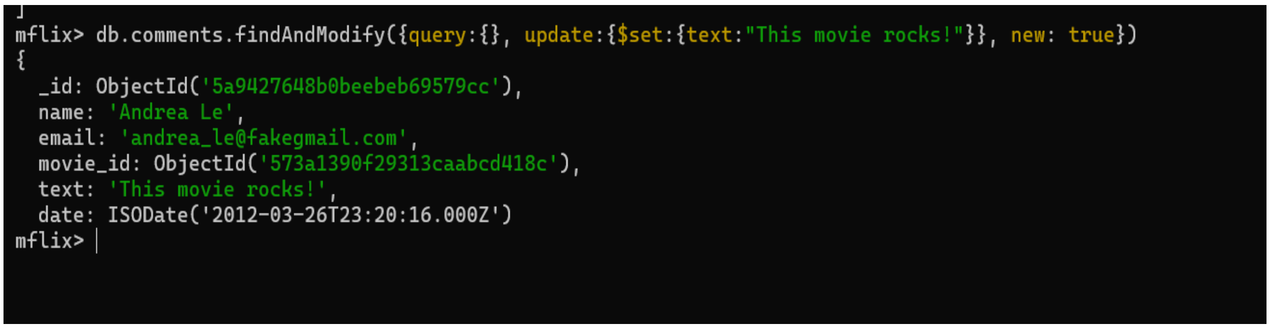
```
  {
    _id: ObjectId('5a9427648b0beebeb69579d3'),
    name: 'Cameron Duran',
    email: 'cameron_duran@fakegmail.com',
    movie_id: ObjectId('573a1390f29313caabcd4217'),
    text: 'Quasi dicta culpa asperiores quaerat perferendis neque. Est animi pariatur impedit itaque exercitationem.',
    date: ISODate('1983-04-27T20:39:15.000Z')
  },
  {
    _id: ObjectId('5a9427648b0beebeb69579d5'),
    name: 'Petyr Baelish',
    email: 'aidan_gillen@gameofthron.es',
    movie_id: ObjectId('573a1390f29313caabcd4218'),
    text: 'Quo deserunt ipsam ipsum. Tenetur eos nemo nam sint praesentium minus exercitationem.',
    date: ISODate('2001-07-13T19:25:09.000Z')
  },
  {
    _id: ObjectId('5a9427648b0beebeb69579d6'),
    name: 'Taylor Hill',
    email: 'taylor_hill@fakegmail.com',
    movie_id: ObjectId('573a1390f29313caabcd4137'),
    text: 'Neque repudiandae laborum earum ipsam facilis blanditiis voluptate. Aliquam vitae porro repellendus voluptatum facere.',
    date: ISODate('1993-10-23T13:16:51.000Z')
  },
  {
    _id: ObjectId('5a9427648b0beebeb69579db'),
    name: 'Olly',
    email: "brenock_o'connor@gameofthron.es",
    movie_id: ObjectId('573a1390f29313caabcd413b'),
    text: 'Perspiciatis sit pariatur quas. Perferendis officia harum ipsum deleniti vel inventore. Nobis culpa eaque in blanditiis porro esse. Nisi deserunt
 culpa expedita dolorum quo aperiam.',
    date: ISODate('2005-01-04T13:49:05.000Z')
  },
  {
    _id: ObjectId('5a9427648b0beebeb69579dd'),
    name: 'Joshua Kent',
    email: 'joshua_kent@fakegmail.com',
    movie_id: ObjectId('573a1390f29313caabcd42ee'),
    text: 'Corporis pariatur rem autem accusamus debitis. Eaque aspernatur quae accusantium non ea quasi ullam. Assumenda quibusdam blanditiis inventore vel
it dolorem. Adipisci quaerat quae architecto sint.',
    date: ISODate('1993-12-06T18:45:21.000Z')
  },
```



mongosh mongodb://127.0.0.

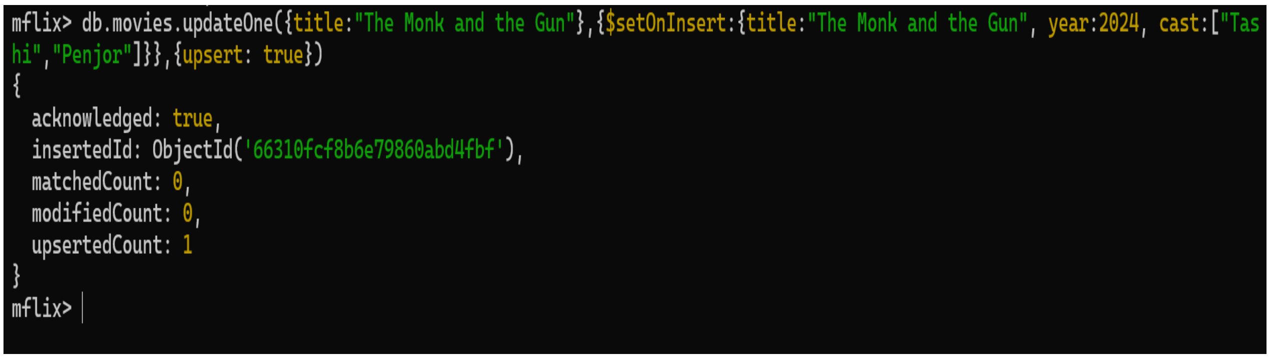
```
    name: 'Daario Naharis',
    email: 'michiel_huisman@gameofthron.es',
    movie_id: ObjectId('573a1390f29313caabcd47c2'),
    text: 'Enim enim deleniti in debitis. Delectus nesciunt id tenetur.',
    date: ISODate('1988-11-19T05:22:18.000Z')
  },
  {
    _id: ObjectId('5a9427648b0beebeb6957a01'),
    name: 'Sarah Lewis',
    email: 'sarah_lewis@fakegmail.com',
    movie_id: ObjectId('573a1390f29313caabcd471c'),
    text: 'Totam molestiae accusamus sed illum aut autem maiores quo. Necessitatibus dolorum sed ea rem. Nihil perferendis fugit tempore quam. Laboriosam al
iquam nulla ratione explicabo unde consectetur.',
    date: ISODate('1981-03-31T06:38:59.000Z')
  },
  {
    _id: ObjectId('5a9427648b0beebeb6957a03'),
    name: 'Beric Dondarrion',
    email: 'richard_dormer@gameofthron.es',
    movie_id: ObjectId('573a1390f29313caabcd47f0'),
    text: 'Placeat sapiente in natus nemo. Qui quibusdam praesentium doloribus aut provident. Optio nihil officia suscipit numquam at.',
    date: ISODate('1998-09-04T04:41:51.000Z')
  },
  {
    _id: ObjectId('5a9427648b0beebeb6957a08'),
    name: 'Meera Reed',
    email: 'ellie_kendrick@gameofthron.es',
    movie_id: ObjectId('573a1390f29313caabcd4964'),
    text: 'Harum porro ad dolorum repellendus. Nihil natus aspernatur quaerat aperiam nam neque. Beatae voluptates quas saepe enim facere. Unde sint praesen
tium numquam molestias nihil.',
    date: ISODate('1971-08-31T07:24:20.000Z')
  },
  {
    _id: ObjectId('5a9427648b0beebeb6957a09'),
    name: 'Daario Naharis',
    email: 'michiel_huisman@gameofthron.es',
    movie_id: ObjectId('573a1390f29313caabcd4aa2'),
    text: 'Distinctio commodi autem amet molestias. Dolorem numquam vitae voluptas corporis fugit aut autem earum.',
    date: ISODate('1979-07-06T20:36:21.000Z')
  }
]
Type "it" for more
mflix>
```

3) **findandModify()** - modified the first comment to say "The movie rocks!".



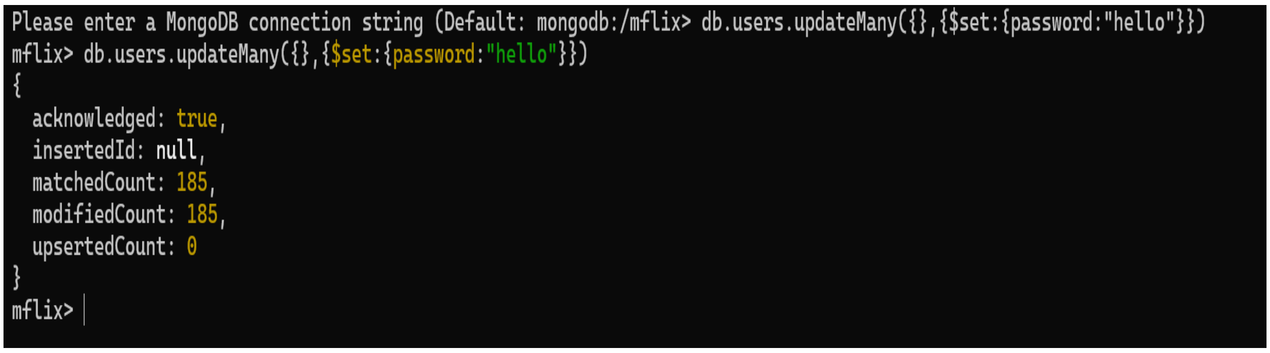
```
]
mflix> db.comments.findAndModify({query:{}, update:{$set:{text:"This movie rocks!"}}, new: true})
{
  _id: ObjectId('5a9427648b0beebeb69579cc'),
  name: 'Andrea Le',
  email: 'andrea_le@fakegmail.com',
  movie_id: ObjectId('573a1390f29313caabcd418c'),
  text: 'This movie rocks!',
  date: ISODate('2012-03-26T23:20:16.000Z')
mflix> 
```

4) **updateOne()** - updated the one movie "The Monk and the Gun" and I was successfully able to add a new movie since the movie doesn't exist in the database.



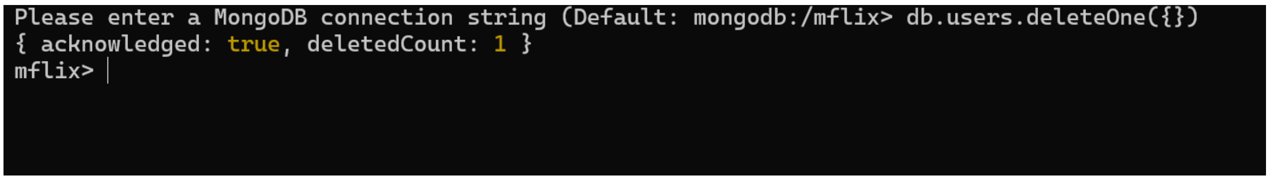
```
mflix> db.movies.updateOne({title:"The Monk and the Gun"},{$setOnInsert:{title:"The Monk and the Gun", year:2024, cast:["Tas
hi","Penjor"]}},{upsert: true})
{
  acknowledged: true,
  insertedId: ObjectId('66310fcf8b6e79860abd4fbf'),
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 1
}
mflix> 
```

5) **updateMany()** - updating all the users(185 users) password to "hello".



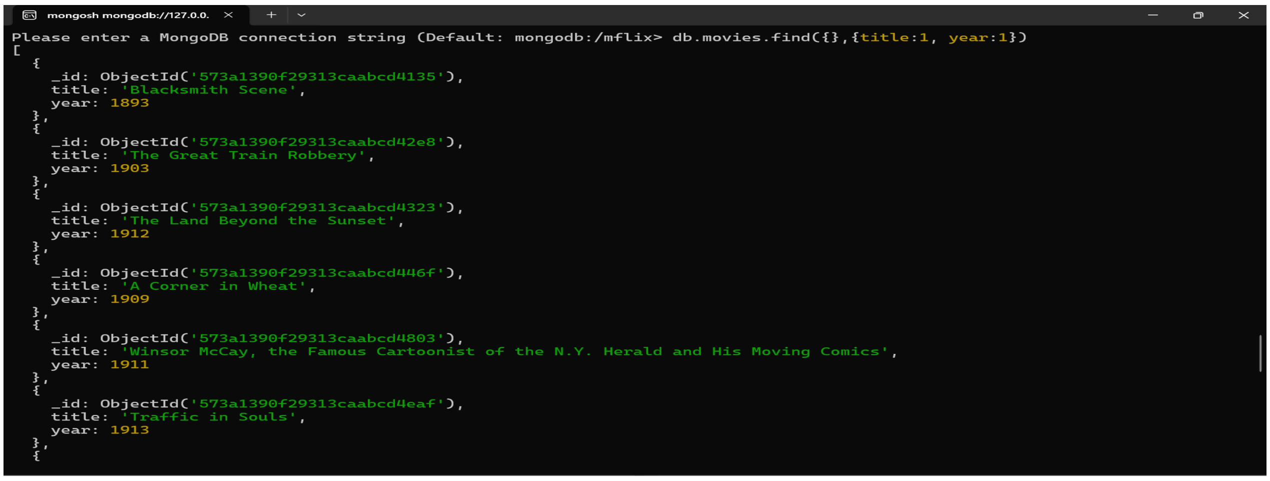
```
Please enter a MongoDB connection string (Default: mongodb:/mflix> db.users.updateMany({},{$set:{password:"hello"}})
mflix> db.users.updateMany({},{$set:{password:"hello"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 185,
  modifiedCount: 185,
  upsertedCount: 0
}
mflix> 
```

6) **deleteOne()** - deleting one user. Since then I deleted the first user from the sample database.

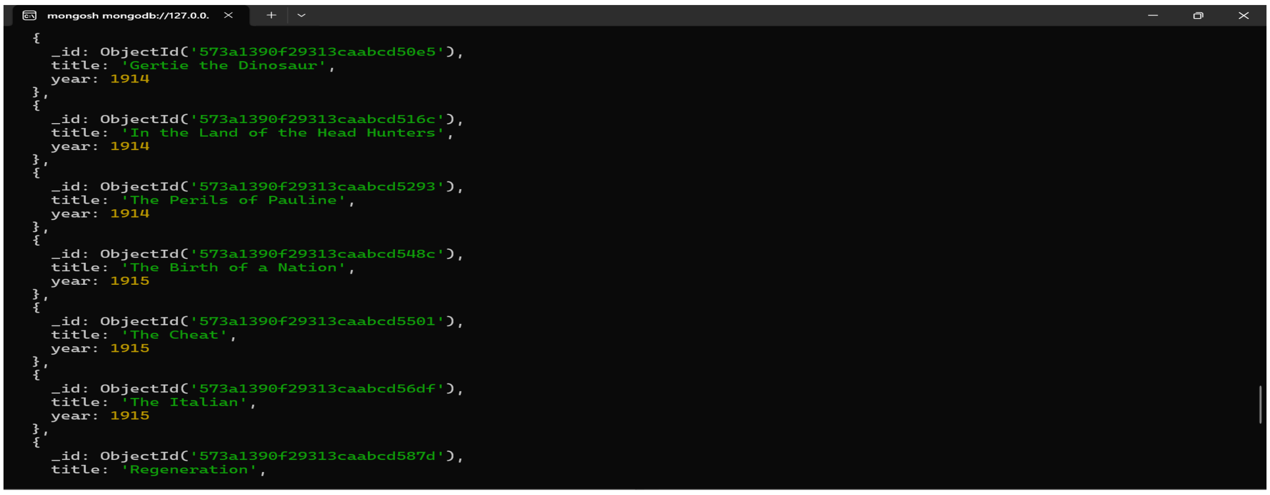


```
Please enter a MongoDB connection string (Default: mongodb:/mflix> db.users.deleteOne({})
{ acknowledged: true, deletedCount: 1 }
mflix>
```

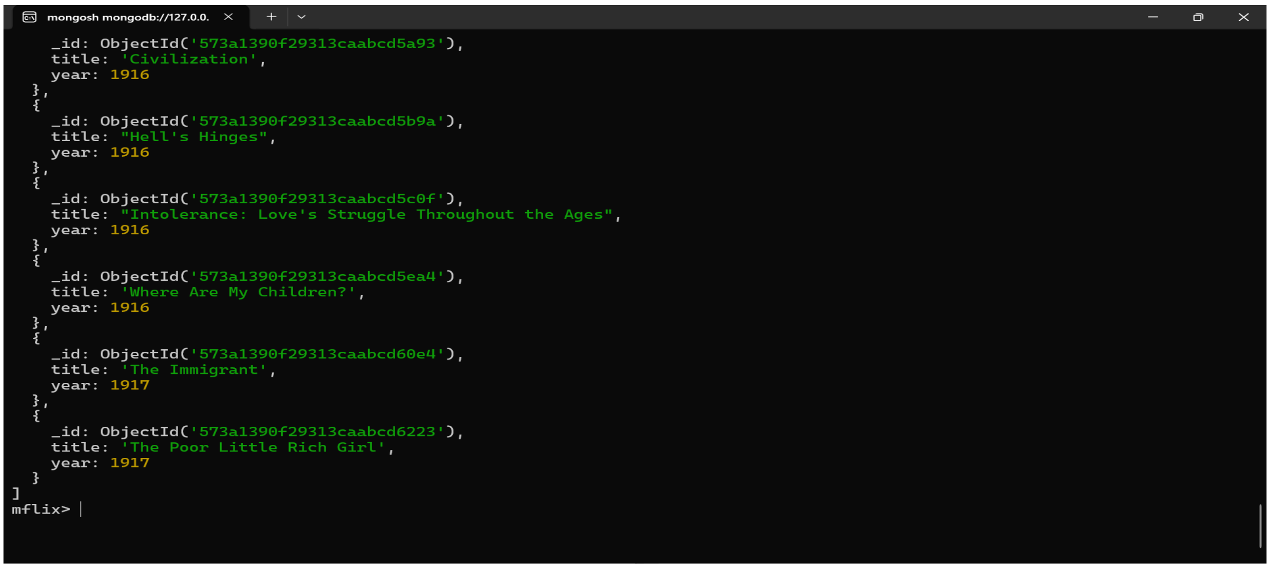
7) **projection :** I used the find method to apply the projection in the movies to find the particular array element in the movies.



```
Please enter a MongoDB connection string (Default: mongodb:/mflix> db.movies.find({},{title:1, year:1})
[
  {
    _id: ObjectId('573a1390f29313caabcd4135'),
    title: 'Blacksmith Scene',
    year: 1893
  },
  {
    _id: ObjectId('573a1390f29313caabcd42e8'),
    title: 'The Great Train Robbery',
    year: 1903
  },
  {
    _id: ObjectId('573a1390f29313caabcd4323'),
    title: 'The Land Beyond the Sunset',
    year: 1912
  },
  {
    _id: ObjectId('573a1390f29313caabcd446f'),
    title: 'A Corner in Wheat',
    year: 1909
  },
  {
    _id: ObjectId('573a1390f29313caabcd4803'),
    title: 'Winsor McCay, the Famous Cartoonist of the N.Y. Herald and His Moving Comics',
    year: 1911
  },
  {
    _id: ObjectId('573a1390f29313caabcd4eaf'),
    title: 'Traffic in Souls',
    year: 1913
  },
  {
```

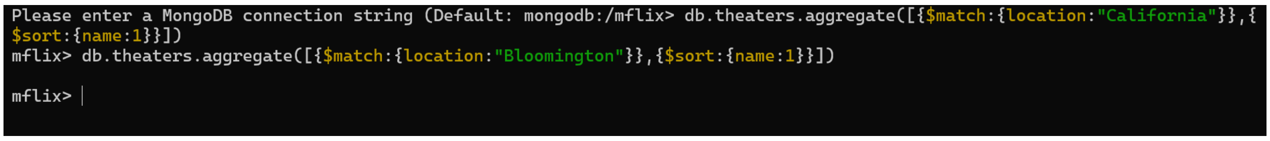


```
  {
    _id: ObjectId('573a1390f29313caabcd50e5'),
    title: 'Gertie the Dinosaur',
    year: 1914
  },
  {
    _id: ObjectId('573a1390f29313caabcd516c'),
    title: 'In the Land of the Head Hunters',
    year: 1914
  },
  {
    _id: ObjectId('573a1390f29313caabcd5293'),
    title: 'The Perils of Pauline',
    year: 1914
  },
  {
    _id: ObjectId('573a1390f29313caabcd548c'),
    title: 'The Birth of a Nation',
    year: 1915
  },
  {
    _id: ObjectId('573a1390f29313caabcd5501'),
    title: 'The Cheat',
    year: 1915
  },
  {
    _id: ObjectId('573a1390f29313caabcd56df'),
    title: 'The Italian',
    year: 1915
  },
  {
    _id: ObjectId('573a1390f29313caabcd587d'),
    title: 'Regeneration',
```



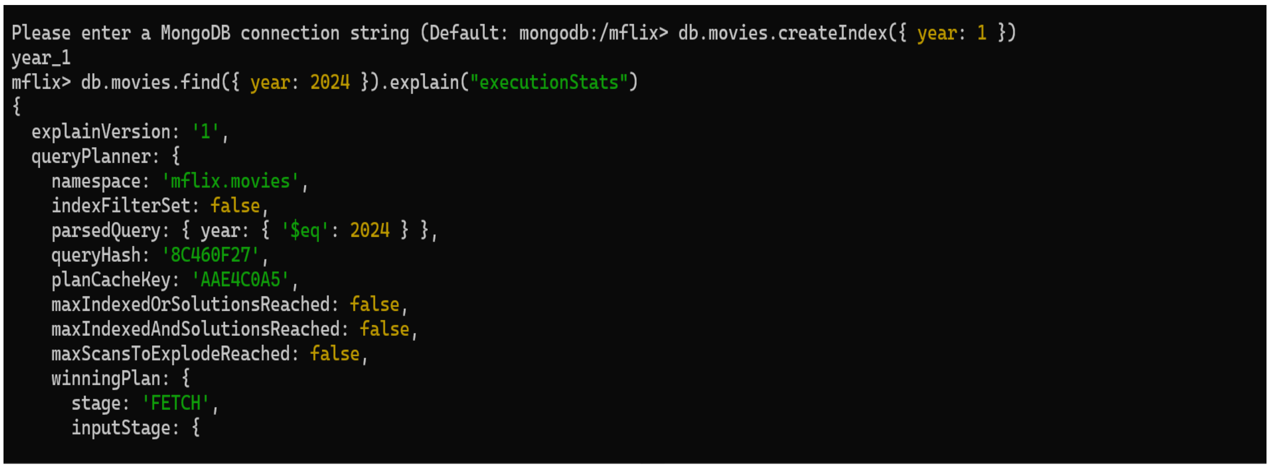
```
mongosh mongodb://127.0.0.
    _id: ObjectId('573a1390f29313caabcd5a93'),
    title: 'Civilization',
    year: 1916
  },
  {
    _id: ObjectId('573a1390f29313caabcd5b9a'),
    title: "Hell's Hinges",
    year: 1916
  },
  {
    _id: ObjectId('573a1390f29313caabcd5c0f'),
    title: "Intolerance: Love's Struggle Throughout the Ages",
    year: 1916
  },
  {
    _id: ObjectId('573a1390f29313caabcd5ea4'),
    title: 'Where Are My Children?',
    year: 1916
  },
  {
    _id: ObjectId('573a1390f29313caabcd60e4'),
    title: 'The Immigrant',
    year: 1917
  },
  {
    _id: ObjectId('573a1390f29313caabcd6223'),
    title: 'The Poor Little Rich Girl',
    year: 1917
  }
]
mflix>
```

8) **aggregation** - it is used to retrieve a list of theaters sorted by name(ascending) located in Bloomington.

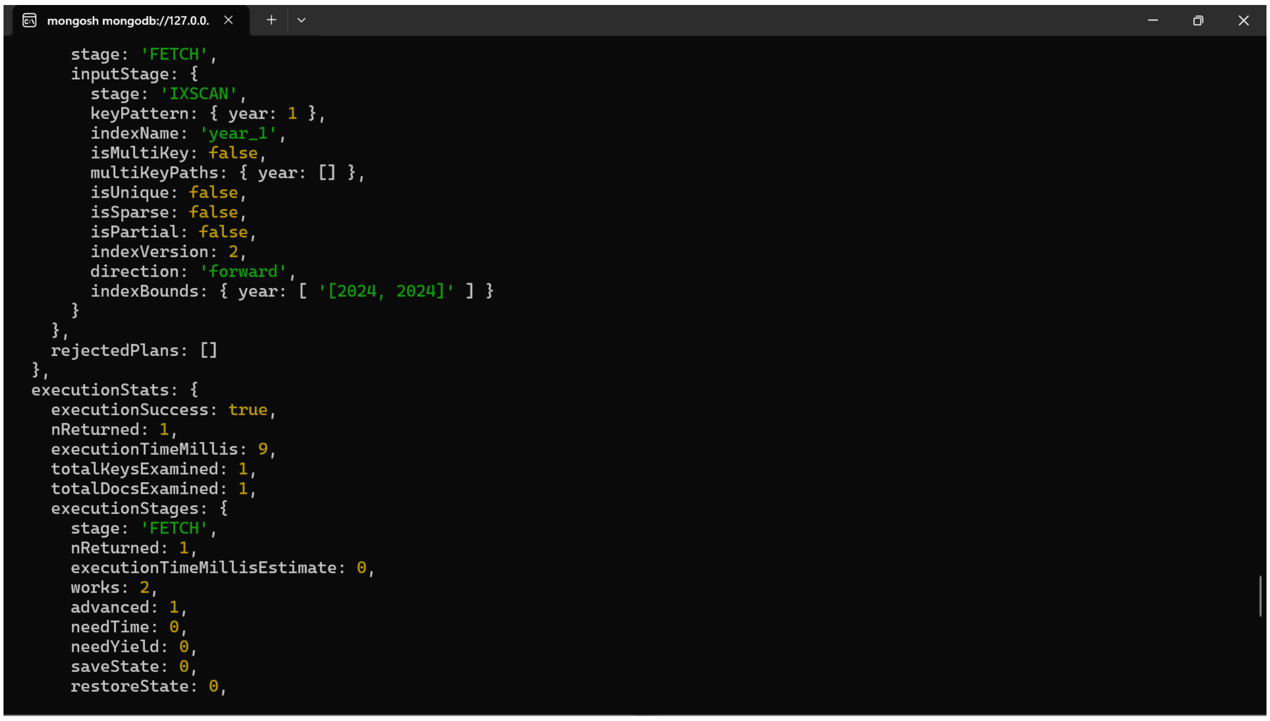


```
Please enter a MongoDB connection string (Default: mongodb:/mflix> db.theaters.aggregate([{$match:{location:"California"}},{
$sort:{name:1}}])
mflix> db.theaters.aggregate([{$match:{location:"Bloomington"}},{$sort:{name:1}}])

mflix>
```

9) **Indexing -** to create the index I queried the movies by year and created the index on the year and to check if the index is being used we can use the explain() method.

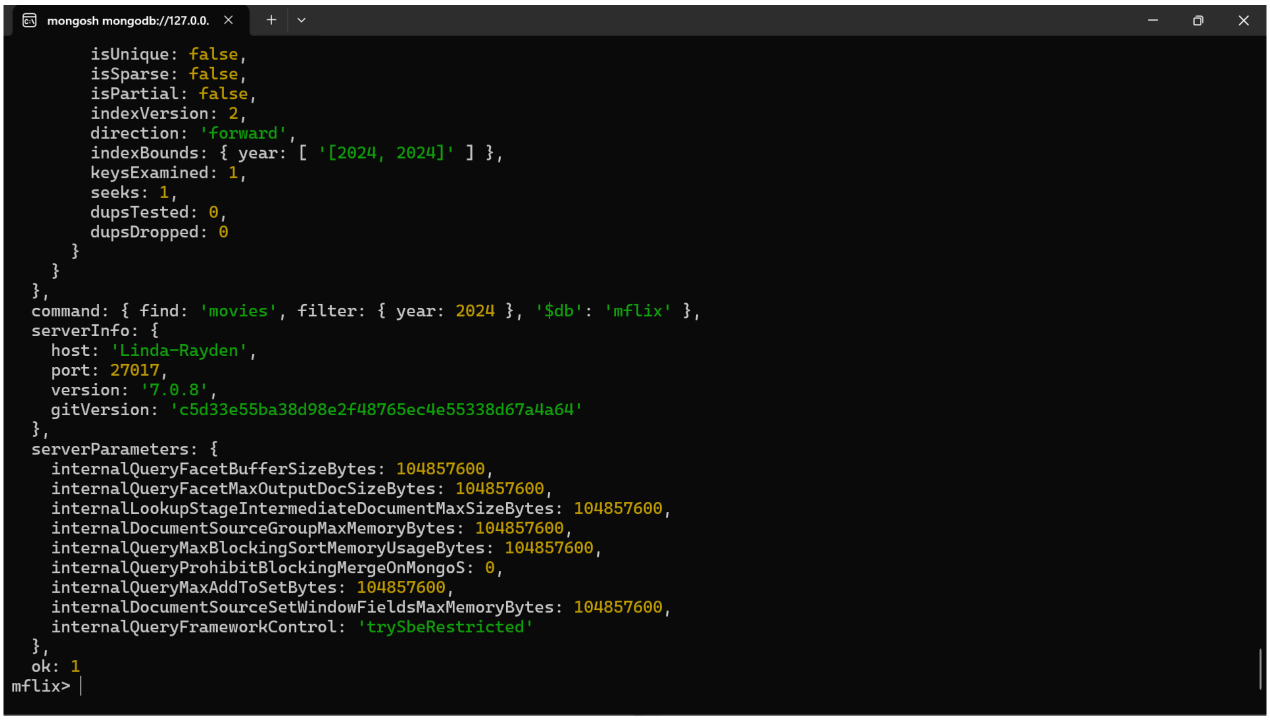


```
Please enter a MongoDB connection string (Default: mongodb:/mflix> db.movies.createIndex({ year: 1 })
year_1
mflix> db.movies.find({ year: 2024 }).explain("executionStats")
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'mflix.movies',
    indexFilterSet: false,
    parsedQuery: { year: { '$eq': 2024 } },
    queryHash: '8C460F27',
    planCacheKey: 'AAE4C0A5',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'FETCH',
      inputStage: {
```



mongosh mongodb://127.0.0.

```
      stage: 'FETCH',
      inputStage: {
        stage: 'IXSCAN',
        keyPattern: { year: 1 },
        indexName: 'year_1',
        isMultiKey: false,
        multiKeyPaths: { year: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { year: [ '[2024, 2024]' ] }
      }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 1,
    executionTimeMillis: 9,
    totalKeysExamined: 1,
    totalDocsExamined: 1,
    executionStages: {
      stage: 'FETCH',
      nReturned: 1,
      executionTimeMillisEstimate: 0,
      works: 2,
      advanced: 1,
      needTime: 0,
      needYield: 0,
      saveState: 0,
      restoreState: 0,
```



mongosh mongodb://127.0.0.

```
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { year: [ '[2024, 2024]' ] },
        keysExamined: 1,
        seeks: 1,
        dupsTested: 0,
        dupsDropped: 0
      }
    }
  },
  command: { find: 'movies', filter: { year: 2024 }, '$db': 'mflix' },
  serverInfo: {
    host: 'Linda-Rayden',
    port: 27017,
    version: '7.0.8',
    gitVersion: 'c5d33e55ba38d98e2f48765ec4e55338d67a4a64'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600,
    internalQueryFrameworkControl: 'trySbeRestricted'
  },
  ok: 1
mflix>
```

Conclusion :

The task performed on the mflix database demonstrates the power of MongoDB for managing and manipulating data by creating a database, downloading and importing a sample dataset. Finally performing various operations such as findOne(), find(),findAndModify(),updateOne(),updateMany(),deleteOne(),projection,aggregation and indexing.

The findOne() and find() operations allowed us to retrieve specific documents or all documents within a collection, showcasing MongoDB's ability to query data efficiently.

The findAndModify() operation, as highlighted in the sources, is particularly useful for updating documents atomically, ensuring data integrity in high-concurrency environments. This operation was successfully used to modify a comment, demonstrating its practical application.

The updateOne() and updateMany() operations were utilized to add a new movie and update user passwords, respectively, illustrating MongoDB's flexibility in handling both single and multiple document updates.

The deleteOne() operation was used to remove a user, demonstrating MongoDB's capability to delete documents based on specified criteria.

Projection was applied to retrieve specific fields from documents, highlighting MongoDB's support for aggregation expressions and syntax in projections.

Aggregation was used to sort theaters by name, showcasing MongoDB's powerful aggregation framework for complex data processing tasks.

Indexing was created on the year field to optimize queries, and the explain() method was used to verify the index's effectiveness, demonstrating MongoDB's support for performance optimization through indexing.