

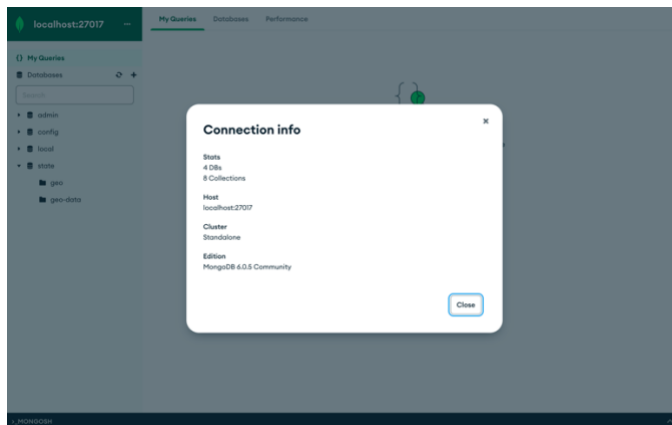
# CIS 552: Database Design – Homework 6

## Introduction:

In order to perform CRUD operations on MongoDB deployments, I am utilizing both the MongoDB Shell and MongoDB Compass. Specifically, I am using the 'mongosh' environment to interact with the database. This setup allows for efficient and effective management of data within the MongoDB environment.

## Pre-Requisites:

### Connected using MongoDB Compass



### Connected using MongoDB Shell:

```
root@7e97e7fcd8e0:/# mongosh --username root --password
Enter password: *****
Current Mongosh Log ID: 6428ebb82520d2b672237bc5
Connecting to: mongosh://<credentials>@127.0.0.1:27017/?directConnection=true&se
rverSelectionTimeoutMS=2000&appName=mongosh+1.8.0
Using MongoDB: 6.0.5
Using Mongosh: 1.8.0

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

-----
The server generated these startup warnings when booting
2023-04-01T19:11:51.325+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
2023-04-01T19:11:52.803+00:00: vm.max_map_count is too low
-----

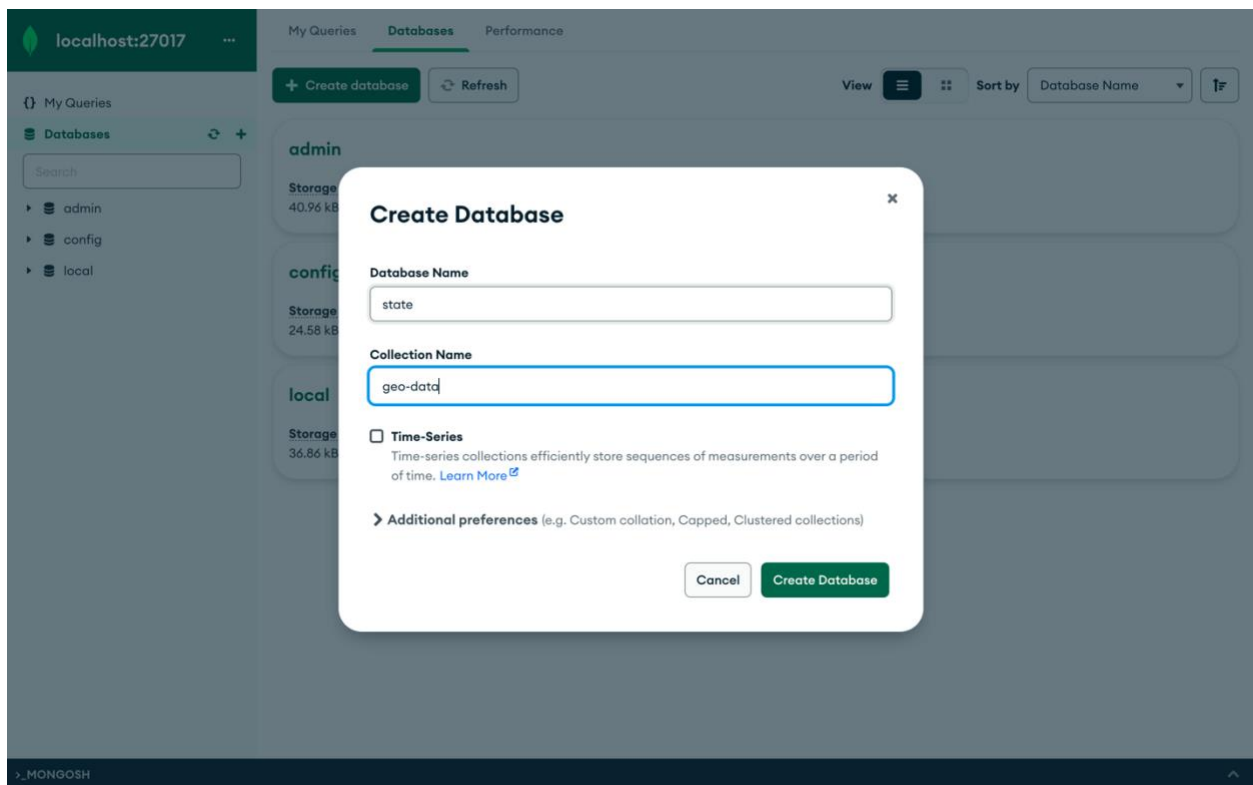
-----
Enable MongoDB's free cloud-based monitoring service, which will then receive and display metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you and anyone you share the URL with. MongoDB may use this information to make product improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
-----
```

Creating a Database:

Using MongoDB Compass:



The above image shows the step to create a database.

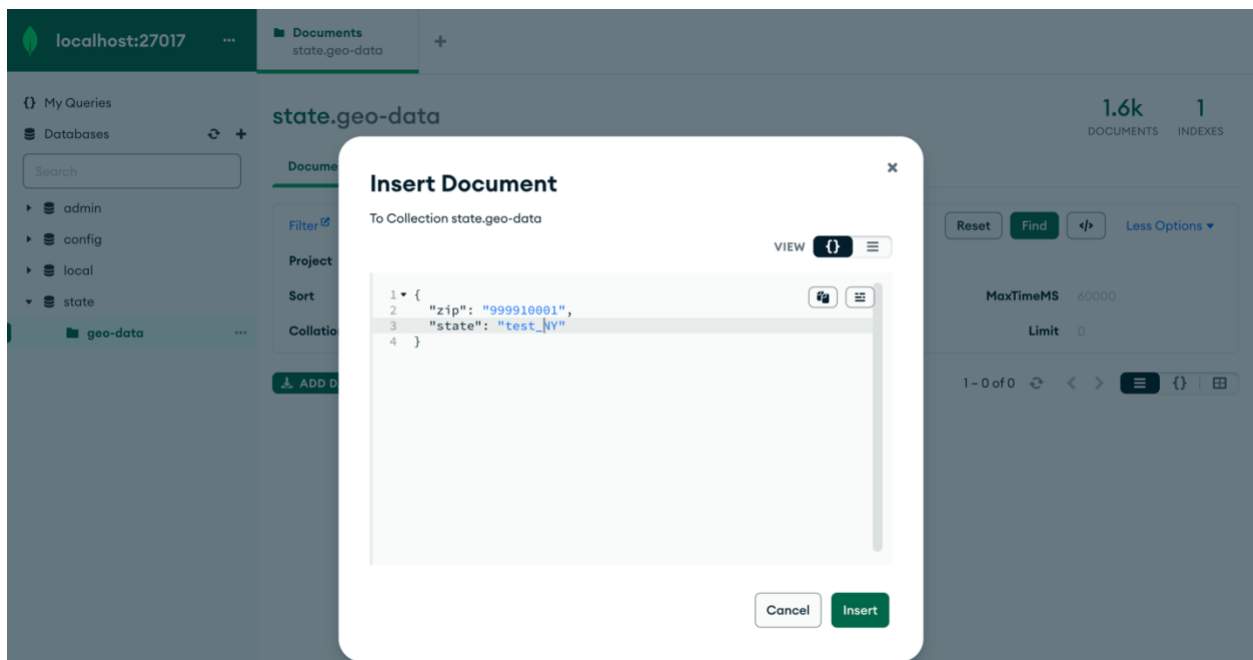
Using MongoDB Shell:

```
test> use state
switched to db state
state> db.geo.insertOne({"zip": "10001", "state": "NY"});
{
  acknowledged: true,
  insertedId: ObjectId("6428ec22a771a90d269cee47")
}
```

In MongoDB Shell, inserting a data will create a collection if not exists.

## Inserting Document:

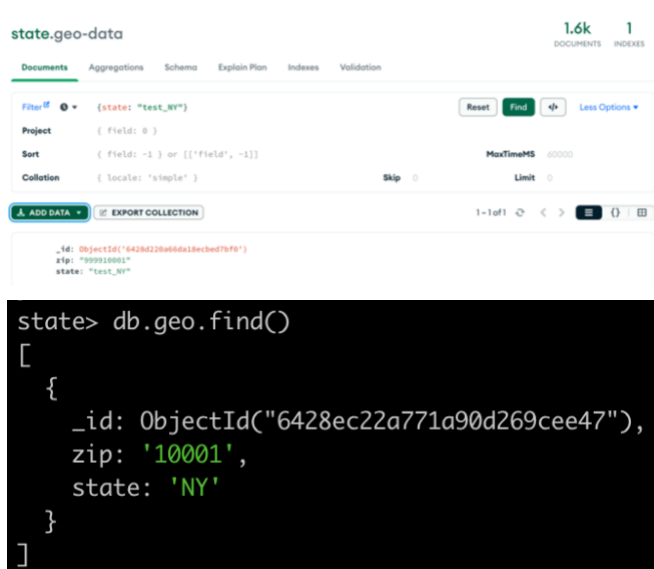
Using MongoDB Compass:



Using MongoDB Shell:

```
state> db.geo.insertOne({"zip": "10001", "state": "NY"});
{
  acknowledged: true,
  insertedId: ObjectId("6428ec22a771a90d269cee47")
}
```

Acknowledgement:



```
state> db.geo.find()
[
  {
    _id: ObjectId("6428ec22a771a90d269cee47"),
    zip: '10001',
    state: 'NY'
  }
]
```

## Updating Document:

### Using MongoDB Compass:

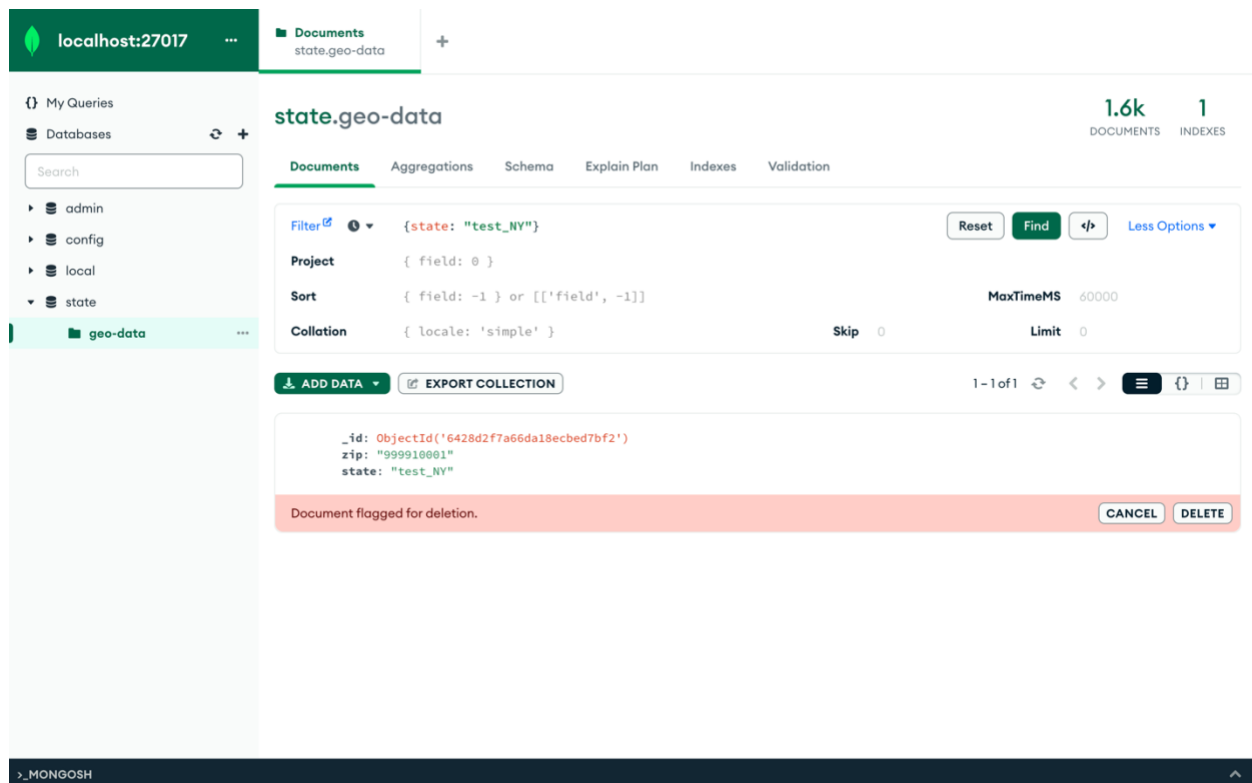
The screenshot shows the MongoDB Compass web interface. On the left, a sidebar lists databases: admin, config, local, and state. The 'state' database is selected, and the 'geo-data' collection is highlighted. The main panel displays the 'state.geo-data' collection with 1.6k documents and 1 index. The 'Documents' tab is active, showing a filter of `{state: "test_NY"}`. Below the filter, the document structure is shown: `Project: { field: 0 }`, `Sort: { field: -1 } or [['field', -1]]`, and `Collation: { locale: 'simple' }`. The document is displayed in a table with columns: `_id` (ObjectId), `zip` (String), and `state` (String). The current document has `zip: '10001'` and `state: 'test_NY'`. A yellow banner at the bottom indicates 'Document modified.' with 'CANCEL' and 'UPDATE' buttons.

### Using MongoDB Shell:

```
state> db.geo.updateOne({state: 'NY'},{$set: {zip: '10002'}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
state> db.geo.find()
[
  {
    _id: ObjectId("6428ec22a771a90d269cee47"),
    zip: '10002',
    state: 'NY'
  }
]
```

## Delete Documents:

### Using MongoDB Compass:



### Using MongoDB Shell:

```
state> db.geo.find({zip: '10002'})
[
  {
    _id: ObjectId("6428ec22a771a90d269cee47"),
    zip: '10002',
    state: 'NY'
  }
]
state> db.geo.deleteOne({zip: '10002'})
{ acknowledged: true, deletedCount: 1 }
state> db.geo.find({zip: '10002'})

state> 
```

## Query Documents:

### Using MongoDB Compass:

The screenshot shows the MongoDB Compass web interface. On the left, a sidebar lists databases: 'admin', 'config', 'local', and 'state'. The 'state' database is expanded, showing a collection named 'geo-data'. The main panel displays the 'state.geo-data' collection with 1.6k documents and 1 index. The 'Documents' tab is active, showing a filter of `{state: "test_NY"}`. Below the filter, the 'Project' section shows `{ field: 0 }`, the 'Sort' section shows `{ field: -1 } or [['field', -1]]`, and the 'Collation' section shows `{ locale: 'simple' }`. The 'Skip' value is 0 and the 'Limit' is 0. A 'Find' button is visible. Below the query settings, there are buttons for 'ADD DATA' and 'EXPORT COLLECTION'. At the bottom, a document is displayed: `{ _id: ObjectId('6428d220a66da18ecbed7bf0'), zip: "999910001", state: "test_NY" }`. The bottom status bar shows '>\_MONGOSH'.

### Using MongoDB Shell:

```
state> db.geo.find({zip: '10001'})
[
  {
    _id: ObjectId("6428f0ada771a90d269cee48"),
    zip: '10001',
    state: 'NY'
  }
]
```

## Import Collection:

The screenshot shows the MongoDB Compass interface with the 'state.geo-data' collection selected. An 'Import' dialog box is open, indicating the file 'zip-state.json' is being imported into the collection. The dialog shows a progress bar and the message 'Import completed 1,596 / 1,596'. The background shows the collection's documents, including fields like '\_id', 'zip', and 'state'.

localhost:27017 Documents state.geo-data

My Queries Databases Search

admin config local state geo geo-data

state.geo-data 1.6k DOCUMENTS 1 INDEXES

Documents Aggregations Schema Explain Plan Indexes Validation

Filter Filter

ADD DATA

Import

To Collection state.geo-data

Select File

zip-state.json

☐ Stop on errors

Import completed 1,596 / 1,596

Done

Reset Find More Options

1 - 20 of 1596

\_id: ObjectId('5c8eccc1ca187d17ca72f8b')  
zip: "10003"  
state: "NY"

\_id: ObjectId('5c8eccc1ca187d17ca72f8c')  
zip: "10004"  
state: "NY"

>\_MONGOSH

## Imported Data:

The screenshot shows the MongoDB Compass interface with the 'state.geo-data' collection selected. The 'Documents' tab is active, displaying a list of documents. The documents contain fields like '\_id', 'zip', and 'state'. The interface shows 1.6k documents and 1 index.

localhost:27017 Documents state.geo-data

My Queries Databases Search

admin config local state geo geo-data

state.geo-data 1.6k DOCUMENTS 1 INDEXES

Documents Aggregations Schema Explain Plan Indexes Validation

Filter Filter Type a query: { field: 'value' }

Reset Find More Options

ADD DATA EXPORT COLLECTION

1 - 20 of 1596

\_id: ObjectId('5c8eccc1ca187d17ca72f8a')  
zip: "10001"  
state: "NY"

\_id: ObjectId('5c8eccc1ca187d17ca72f8d')  
zip: "10005"  
state: "NY"

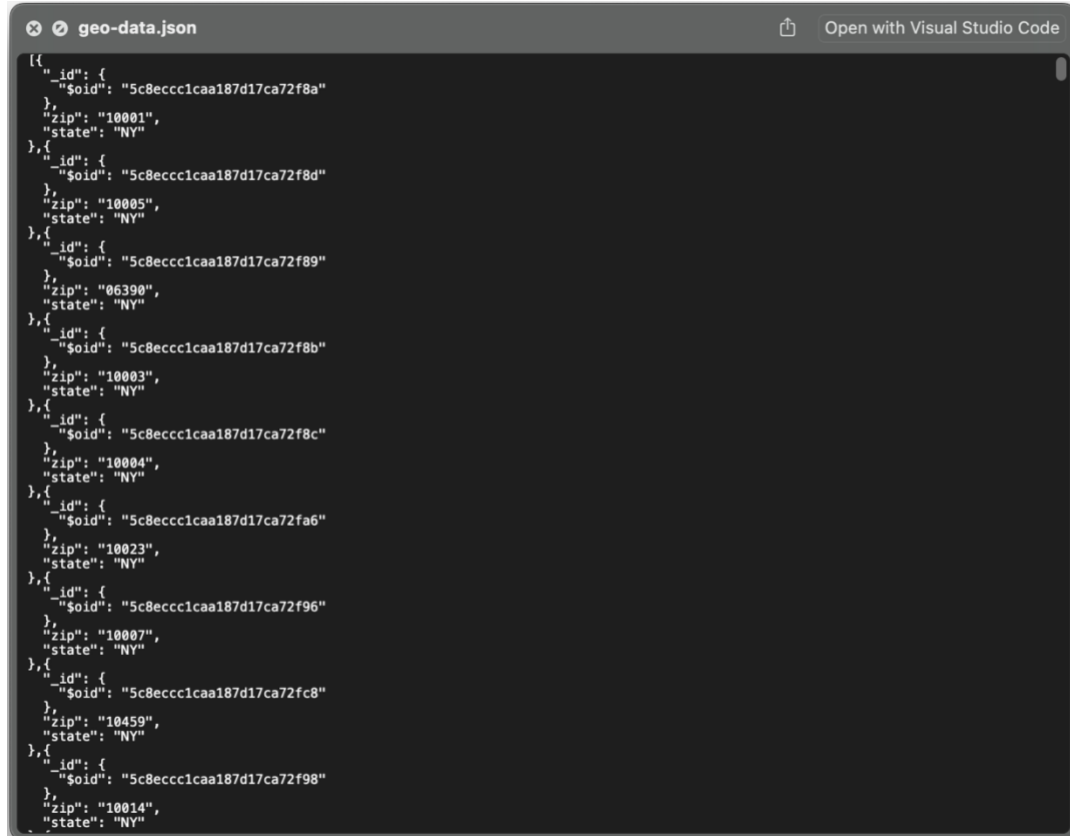
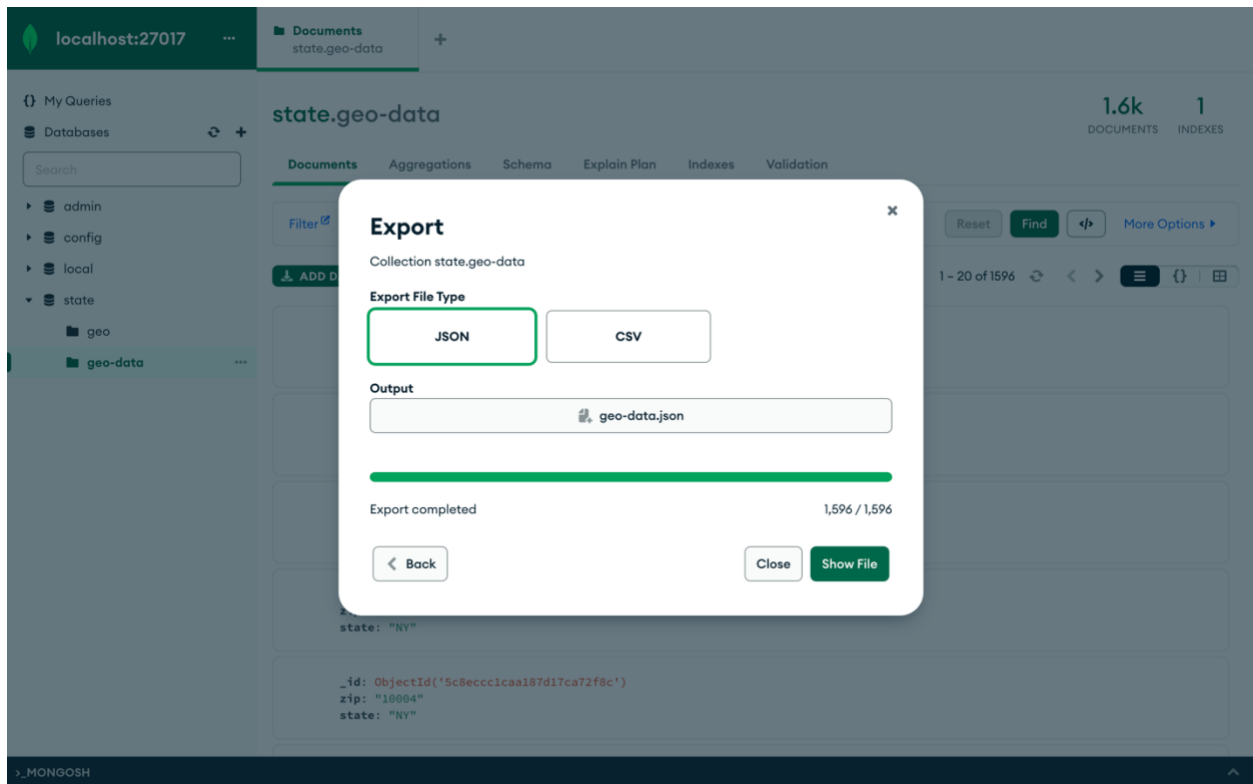
\_id: ObjectId('5c8eccc1ca187d17ca72f89')  
zip: "06390"  
state: "NY"

\_id: ObjectId('5c8eccc1ca187d17ca72f8b')  
zip: "10003"  
state: "NY"

\_id: ObjectId('5c8eccc1ca187d17ca72f8c')  
zip: "10004"  
state: "NY"

>\_MONGOSH

## Exporting Collection:





## Conclusion:

In conclusion, this assignment has provided hands-on experience with MongoDB, a popular NoSQL database system. Through the use of tools such as MongoDB Shell and MongoDB Compass, CRUD operations were performed, allowing for efficient management of data within MongoDB deployments. The 'mongosh' environment was particularly useful for interacting with the database. Additionally, MongoDB Compass was used to import and export collections, allowing for seamless data transfer between databases and backup of important data. This feature adds another layer of functionality to MongoDB Compass, making it a valuable tool in database management.