Southern New Hampshire University

CS-340: Assignment 3-1: Module 3 Journal

Prepared on: September 24, 2025
Prepared for: Prof. Jeff H. Sanford
Prepared by: Alexander Ahmann

Contents

1 Introduction

This writeup, along with previous ones to an extent, represents work that culminates into the first project.[1] In the last two writeups, I document my experiences working with importing JSON data into a MongoDB instance, modelling their datasets, and manipulating them with *create*, *read*, *update* and *delete* (CRUD) queries.

In this current writeup, I will document my work in creating an arbitrary single-field index, an arbitrary compound index with a partial filter expression, and then perform basic MongoDB systems administration tasks involving *role-based access controls* (RoBAC).

1.1 Brief description of the figures

Figures depicting screenshots demonstrating task completion are shown in Appendix B. In order to demonstrate that the given screenshots in the appendix are my original work, I included my *Codio* username `AAHMANN` in the screenshots. Furthermore, when taking screenshots, I include the errors and warnings for the sake of a more transparent disclosure, and to hopefully make them "stand out" more as original work.

- Figures 1 and 2 depict screenshots of importing the CSV dataset into the *MongoDB* instance. Specifically, figure 1 shows the application of the `mongoimport` utility to import the CSV file and logging into the *MongoDB* instance with `mongosh` utility. Figure 2 shows a small sampling of the documents from the CSV import.

- Figures 3, 4 and 5 depict the creation of a simple, single-field index. Specifically, Figure 3 shows the creation of a single-field index and sampling documents `animals` collection, figure 4 shows a brief example of using the `explain()` function on that same sample, and figure 5 applies the `explain()` function to a `find()` query using a `hint()` involving the `breed_1` index.

- Figures 6 and 7 depict the creation of a compound index. Specifically, figure 6 shows the creation of a compound-index, involving a partial-field index selecting `outcome_index` with values of `Transfer`. Figure 7

[1] See Prof. Sanford (Sept. 15, 2025). *Welcome to Module 3.*

shows the `explain()` function applied to a `find` query with a `hint()` on the `outcome_index_1` index.

- Figures 8 and 9 depict adding a new user to the *MongoDB* instance, and logging in as the new user. Specifically, figure 8 shows the creation of a new user `aacuser` with `readWrite` permissions on the `aac` database, and figure 9 depicts logging into the *MongoDB* instance under the `aacuser` account, and querying its permissions.

2 Problem Set - Part I: MongoDB Indexes

2.1 Task 1.1: Importing a CSV dataset

> "In Codio, open the terminal window to access the Linux shell. Upload the *Austin Animal Center* (AAC) Outcomes data set into MongoDB by importing a CSV file using the appropriate MongoDB import tool. Use the database name `aac` and collection name `animals`. Complete the import using the `mongoimport` tool, and take screenshots of both the import command and its execution."

Unlike in previous assignments,[2] the student is left to their understanding of the `mongoimport` tool and their imagination to import the given CSV dataset into the MongoDB instance.[3] From previous experience and online documentation, I have worked out that the proper syntax for having `mongoimport` import a CSV database is shown in table 1.

Table 1:

```
mongoimport --db <database name>
  --collection <collection name>
  --type csv
  --file [path to CSV file]
  --headerline
```

[2] See Ahmann (2025a,b).

[3] In the form of a "tip," CS-340 (n.d., §1) advises the student to consult the MongoDB documentation for the `mongoimport` utility:

- MongoDB Docs (n.d.). *mongoimport*. Last retrieved on Sept. 16, 2025 from: https://www.mongodb.com/docs/database-tools/mongoimport/

The database name is to be `aac`, the collection name is to be `animals`, and the target CSV file is called `aac_shelter_outcomes.csv`.[4] The exact `mongoimport` command that I used to import that CSV file into the *MongoDB instance* is shown in table 2.

Table 2:

```
mongoimport --db aac
  --collection animals
  --type csv
  --file ./aac_shelter_outcomes.csv
  --headerline
```

I then ran `mongosh` to interact with the *MongoDB* instance, and ran the commands `show dbs` and `show collections` to confirm that the import was successful. Figure 1 depicts a screenshot that demonstrates the execution of this task, and their respective results. I also ran `db.animals.find()` to get a small sample of documents in the database, with figure 2 depicting its results.

2.2 Task 1.2: The single-field index, and documenting its performance optimization

> "After importing your data set, open the mongo shell. Create a simple index on the key `breed`. Show an example query using this index, and use the `explain` function to verify that the index will be used. Take screenshots of your example query." — CS-340 (n.d., §2)

An *index* is an application of the B-tree data structure to enable quicker lookups and data retrieval with databases.[5] In this task, I will create a

[4] `aac_shelter_outcomes.csv` is stored in the `/home/codio/workspace/datasets` directory, and I had to `cd` into that directory before executing the `mongoimport` command. Also, the `--headerline` field instructs `mongoimport` to use the columnns in the first row of the CSV file as labels for the fields.

[5] Definition of "index" paraphrased from Giamas (2022, §8.0: Chapter on Indexing's introduction). I also recommend the following resource when learning how to work with indexes:

- MongoDB Docs (n.d.). *Indexes*. Last retrieved on Sept. 17, 2025 from: https://

"simple," single-field index, which is made up of the unique `ObjectId(_id)` data type.[6] The exact MongoDB query that I used to create this simple index of the `breed` field is shown in table 3.

Table 3:

```
db.animals.createIndex({
  "breed": 1
})
```

After executing the query, `breed_1` was printed onto the terminal— which I have worked out is the index's label. I then used the `db.animals.find()` query to get a quick sample of the `animals` collection. Figure 3 depicts the results of this transaction.

Objects in a MongoDB collection come with an `explain()` function that gives a basic description of its properties and, in the case of indexes, details how they optimize performance.[7] To demonstrate its basic usage, I used it to get basic information about the `aac.animals` collection. Figure 4 depicts a screenshot showing its results.

Of course, the task is instructing students to use the `explain` function on the newly created index. I started by using `breed_1`'s `explain()` function, and then selecting for documents that have a key `breed` with the value `Domestic Medium Hair Mix` to get their `queryPlanner` information.[8] The exact queries that I used to perform these tasks are shown in tables 4 and 5.

www.mongodb.com/docs/manual/indexes/

- MongoDB Docs (n.d.). *Single Field Indexes*. Last retrieved on Sept. 17, 2025 from: https://www.mongodb.com/docs/manual/core/indexes/index-types/index-single/

[6]Description of the "single-field index" is paraphrased from Giamas (2022, §8.1: Index types).

[7]My description might be a bit oversimplistic; I recommend the following resources to get a better understanding of the `explain` function:

- MongoDB Docs (n.d.). *explain (database command)*. Last retrieved on Sept. 17, 2025 from: https://www.mongodb.com/docs/manual/reference/command/explain/

[8]See Giamas (2022, §8.3 - Using indexes effectively) for information on using the given index to select document data from a collection.

Table 4:

```
db.animals.breed_1.explain()
```

Table 5:

```
db.animals.find({
  "breed":"Domestic Medium Hair Mix"
}).hint(
  "breed_1"
).explain(
  "queryPlanner"
)
```

Figure 5 depicts a screenshot demonstrating the execution and results of this task, and appendix A.1 shows the full output of running the latter query. In particular, the `hint()` command instructs the `find` query to use the `breed_1` index for a more efficient database transaction, with its "winning plan" discussed in the `explain("queryPlanner")` output.[9] The second query, in particular, addresses the task as articulated in the assignment guidelines and rubric, specifically as written out in CS-340 (n.d., §2).

2.3 Task 1.3: The compound index, partial index, and documenting its performance optimization

> "Create a compound index that will improve the performance of queries looking for breeds that have an `outcome_type` of `Transfer`. Show an example query using this compound index, and use the explain function to confirm the index will be used. Take screenshots of your example query." —CS-340 (n.d., §1.3)

[9]See the following resource for instructions on using indexes for optimization with the `hint` command:

- MongoDB Docs (n.d.). *cursor.hint() (mongosh method)*. Last retrieved on Sept. 17, 2025 from: https://www.mongodb.com/docs/manual/reference/method/cursor.hint/

Compound indexes can be seen as a "super set" of a single-field index, or a more general form of the latter in which multiple fields are expressed in the former thing being defined. The task wants the student to create a compound index such that documents with the key `outcome_type` set to the respective value of `Transfer`.[10] The exact query that I used to build the compound index on these requirements is depicted in table 6.

Table 6:

```
db.animals.createIndex(
  { "outcome_type":1 },
  { partialFilterExpression: { "outcome_type": "Transfer" }
})
```

This resulted in the compound index with the name `outcome_type_1`. I then used the command `db.animals.outcome_type_1.explain()` to verify that the `outcome_type_1` index could be "explainable." After confirming that it is, I then assessed the effectiveness of this newly created compound index by sampling documents from the database with the `outcome_type` of `Transfer`. The exact query that I used to do this is depicted in table 7.

[10]Regarding working with compound indexes and partial indexes, see Giamas (2022, §8.2 - Index Types) and the following resources:

- MongoDB (n.d.). *Compound Indexes*. Last retrieved on Sept. 17, 2025 from: https://www.mongodb.com/docs/manual/core/indexes/index-types/index-compound/
- MongoDB (n.d.). *Create a Compound Index*. Last retrieved on Sept. 17, 2025 from: https://www.mongodb.com/docs/manual/core/indexes/index-types/index-compound/create-compound-index/
- MongoDB (n.d.). *Partial Indexes*. Last retrieved on Sept. 18, 2025 from: https://www.mongodb.com/docs/manual/core/index-partial/

Table 7:

```
db.animals.find(
  {"outcome_type":"Transfer"}
).hint(
  "outcome_type_1"
).explain(
  "queryPlanner"
)
```

Figure 7 depicts a screenshot demonstrating completion of this task, and appendix A.2 shows the full results of the `explain()` output.

3 Problem Set - Part II: Administration Tasks

3.1 Task 2.1: Adding the new user `aacuser` to MongoDB

> "Create a new user account called `aacuser` with read and write access to the database `aac` in the default `admin` authentication database using the mongo shell. Refer to steps 2 and 3 of the MongoDB *Create a User* tutorial for help with this task. You will need to modify the commands so the account name is `aacuser`. Additional information with respect to user management may be found in the User Management in MongoDB document"—CS-340 (n.d., §2.1).

Role-based access control (RoBAC) is important to implement in production systems.[11] In this case, students are tasked with deploying an account called `aacuser` that will only have access to the `aac` database. To do so, I started by logging into the MongoDB instance with the `mongosh` command, and then switching to the `admin` database context with `use admin`. The

[11] See the following resource to learn more about RoBAC in MongoDB instances:

- MongoDB Docs (n.d.). *Create a User on Self-Managed Deployments.* Last retrieved on Sept. 18, 2025 from: https://www.mongodb.com/docs/v6.0/tutorial/create-users/
- Module 3 Resource (n.d.). *CS 340 User Management in MongoDB.*

exact query that I used to create `aacuser` in accordance with the task description given in CS-340 (n.d., §2.1) is depicted in table 8.

Table 8:

```
db.createUser({
  user: "aacuser",
  pwd: passwordPrompt(),
  roles: [ { role: "readWrite", db: "aac" } ]
})
```

The new user to be created is `aacuser`, which is only granted read/write permissions for the `aac` database. For extra security measures, I set the password `pwd` field to grab a password from the end-user's terminal.[12] Specifically, I used the password: `WingsofRedemption` when setting up the new user account. This transaction was successful — see figure 8.

3.2 Task 2.2: Logging into the MongoDB instance with the `aacuser` user account

> "Take a screenshot of your login process to MongoDB using the mongo shell. Be sure you can access MongoDB and list the databases using the `aacuser` account. This task will verify that your `aacuser` account is working. You should be able to include the login command and connection information in one screenshot." —CS-340 (n.d., §2.2)

To test out the new account, I logged into the mongosh instance with a command depicted by table 9:

Table 9:

```
mongosh
  --username aacuser
  --authenticationDatabase admin
```

[12] As opposed to simply embedding the password into the query, this latter approach may lead to information leakage or some other security issue.

I then ran the query depicted in table 10.

Table 10:

```
db.runCommand({
  connectionStatus:1
})
```

After running this query, the `aacuser` user account seems to be working as intended.[13] Figure 9 depicts a screenshot demonstrating this database transaction and its results.

4 Summary

In this assignment, I began to set up a foundation for the first project.[14] This first project involves writing Python code to interface with a MongoDB instance that handles data for a charity. This assignment "sets the stage" by setting up a database for the charity, creating indexes that will optimize performance, and creating user accounts with appropriate permissions to interface with the database.

Specifically, I was able to demonstrate competence in the topics discussed in this week through the completion of the following tasks:

- I created simple, single-field indexes to create "hints" and various heuristics that optimize performance when performing `find()` queries. I also created partial field, compound indexes that select a subset of documents in the database on the basis that a field matches a specific value. This too will improve performance.
- I used the `hint()` function on `find()` queries to instruct said queries to use the indexes when retrieving data. I also used `explain()` function to describe the `queryPlanner` aspects of the query to show how the `hint()`s will improve performance.

[13] The warning message "[a]ccess control is not enabled for the database. Read and write access to data and configuration is unrestricted" suggests that there may be something wrong with the database management system's role-base access controls. I am not quite sure what the problem is, though.

[14] See CS-340 (n.d.). *CS 340 Project One Guidelines and Rubric.*

- I applied basic principles of *role-based access controls* to set up a MongoDB user account with limited access, and was able to demonstrate logging into the MongoDB instance with the `mongosh` utility through the user account with limited access.

Regarding my work involving role-based access controls, this is an important component in deploying systems into production— especially if they are connected to the internet. There are bad actors on the internet, and poorly-configured instances of a MongoDB instance would allow for an attacker to quickly take it over. Through these access controls, potential risk against a production system can be reduced significantly.

References

Ahmann, A. (2025a). *CS-340: Assignment 2-1: Module 2 Journal.*

Ahmann, A. (2025b). *CS-340: Assignment 1-3: Module 1 Journal.*

CS-340 (n.d.). *Module Three Milestone Guidelines and Rubric.*

Giamas, A. (2022). *Mastering MongoDB 6.x: Expert Techniques to Run High-volume and Fault-tolerant Database Solutions Using MongoDB 6.x.* Birmingham, UK: Packt Publishing. Last retrieved on Sept. 12, 2025 from: https://research.ebsco.com/linkprocessor/plink?id=a5bcc20e-3306-36b5-ad4f-0d0bd1f1567e

A Appendix A: MongoDB Outputs

A.1 Results of "explaining" the queryPlanner of breed_1

```
aac> db.animals.find({
...   "breed":"Domestic Medium Hair Mix"
... }).hint(
...   "breed_1"
... ).explain(
...   "queryPlanner"
... )
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'aac.animals',
    indexFilterSet: false,
    parsedQuery: { breed: { '$eq': 'Domestic Medium Hair Mix' } },
    queryHash: '918BA90C',
    planCacheKey: '0C674662',
    optimizationTimeMillis: 0,
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'FETCH',
      inputStage: {
        stage: 'IXSCAN',
        keyPattern: { breed: 1 },
        indexName: 'breed_1',
        isMultiKey: false,
        multiKeyPaths: { breed: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: {
          breed: [
            '["Domestic Medium Hair Mix",
              "Domestic Medium Hair Mix"]'
          ]
```

```
        }
      }
    },
    rejectedPlans: []
  },
  command: {
    find: 'animals',
    filter: { breed: 'Domestic Medium Hair Mix' },
    hint: 'breed_1',
    '$db': 'aac'
  },
  serverInfo: {
    host: 'welcomeexport-riosavage',
    port: 27017,
    version: '7.0.21',
    gitVersion: 'a47b62aff2bae1914085c3ef1d90fc099acf000c'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600,
    internalQueryFrameworkControl: 'forceClassicEngine'
  },
  ok: 1
}
```

A.2 Results of "explaining" the queryPlanner of outcome_type

```
aac> db.animals.find(
...   {"outcome_type":"Transfer"}
... ).hint(
...   "outcome_type_1"
... ).explain(
...   "queryPlanner"
... )
{
```

```
explainVersion: '1',
queryPlanner: {
  namespace: 'aac.animals',
  indexFilterSet: false,
  parsedQuery: { outcome_type: { '$eq': 'Transfer' } },
  queryHash: '6DA26ACE',
  planCacheKey: '8C3D162C',
  optimizationTimeMillis: 0,
  maxIndexedOrSolutionsReached: false,
  maxIndexedAndSolutionsReached: false,
  maxScansToExplodeReached: false,
  winningPlan: {
    stage: 'FETCH',
    inputStage: {
      stage: 'IXSCAN',
      keyPattern: { outcome_type: 1 },
      indexName: 'outcome_type_1',
      isMultiKey: false,
      multiKeyPaths: { outcome_type: [] },
      isUnique: false,
      isSparse: false,
      isPartial: true,
      indexVersion: 2,
      direction: 'forward',
      indexBounds: { outcome_type: [ '["Transfer", "Transfer"]' ] }
    }
  },
  rejectedPlans: []
},
command: {
  find: 'animals',
  filter: { outcome_type: 'Transfer' },
  hint: 'outcome_type_1',
  '$db': 'aac'
},
serverInfo: {
  host: 'welcomeexport-riosavage',
  port: 27017,
  version: '7.0.21',
  gitVersion: 'a47b62aff2bae1914085c3ef1d90fc099acf000c'
},
```

```
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600,
    internalQueryFrameworkControl: 'forceClassicEngine'
  },
  ok: 1
}
```

B Appendix B: Screenshots

B.1 Figures 1 & 2: Importing the `aac.animals` collection, and sampling its documents

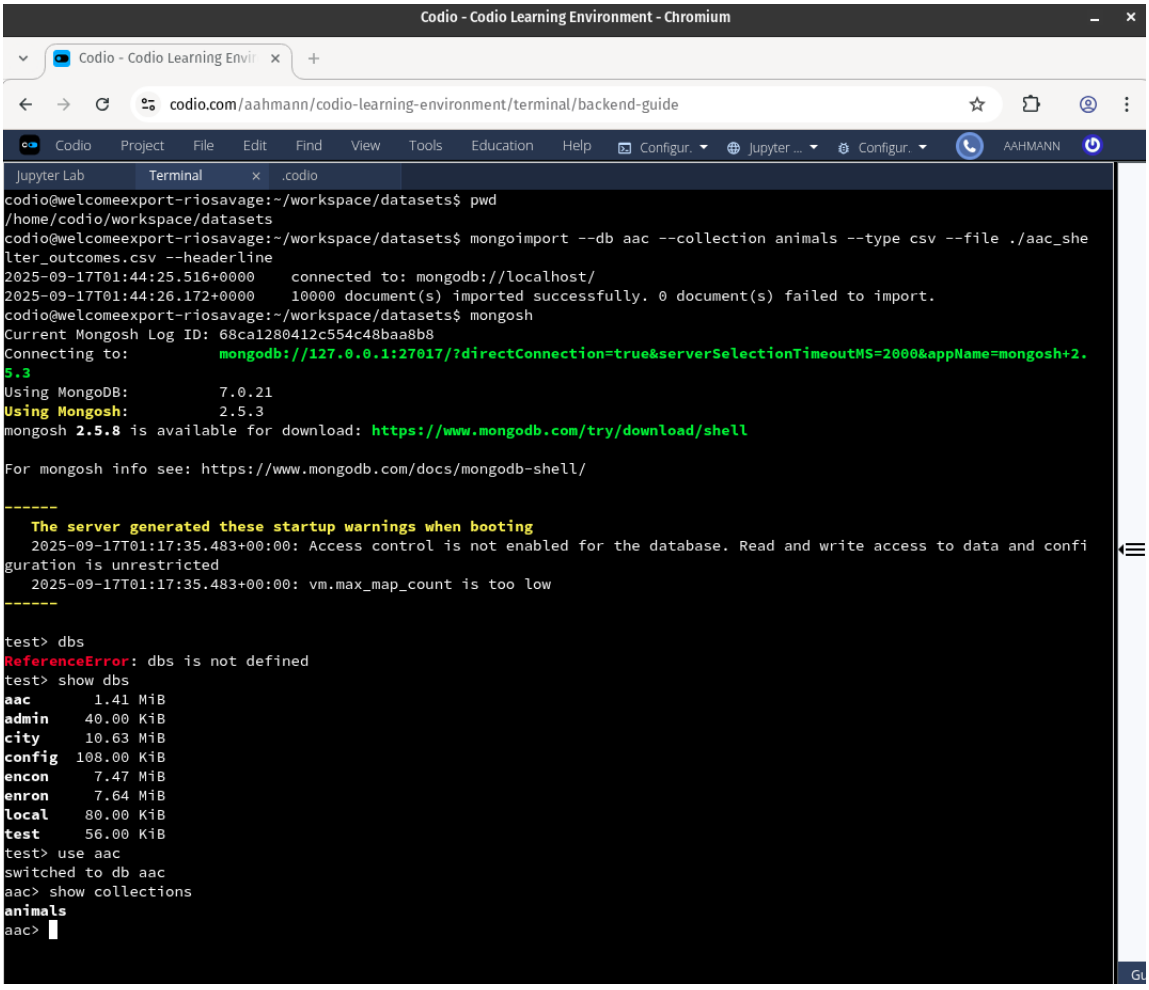

Figure 1: Using `mongoimport` to import the *Austin Animal Center* CSV file, and confirming its existence in the `mongosh` environment.

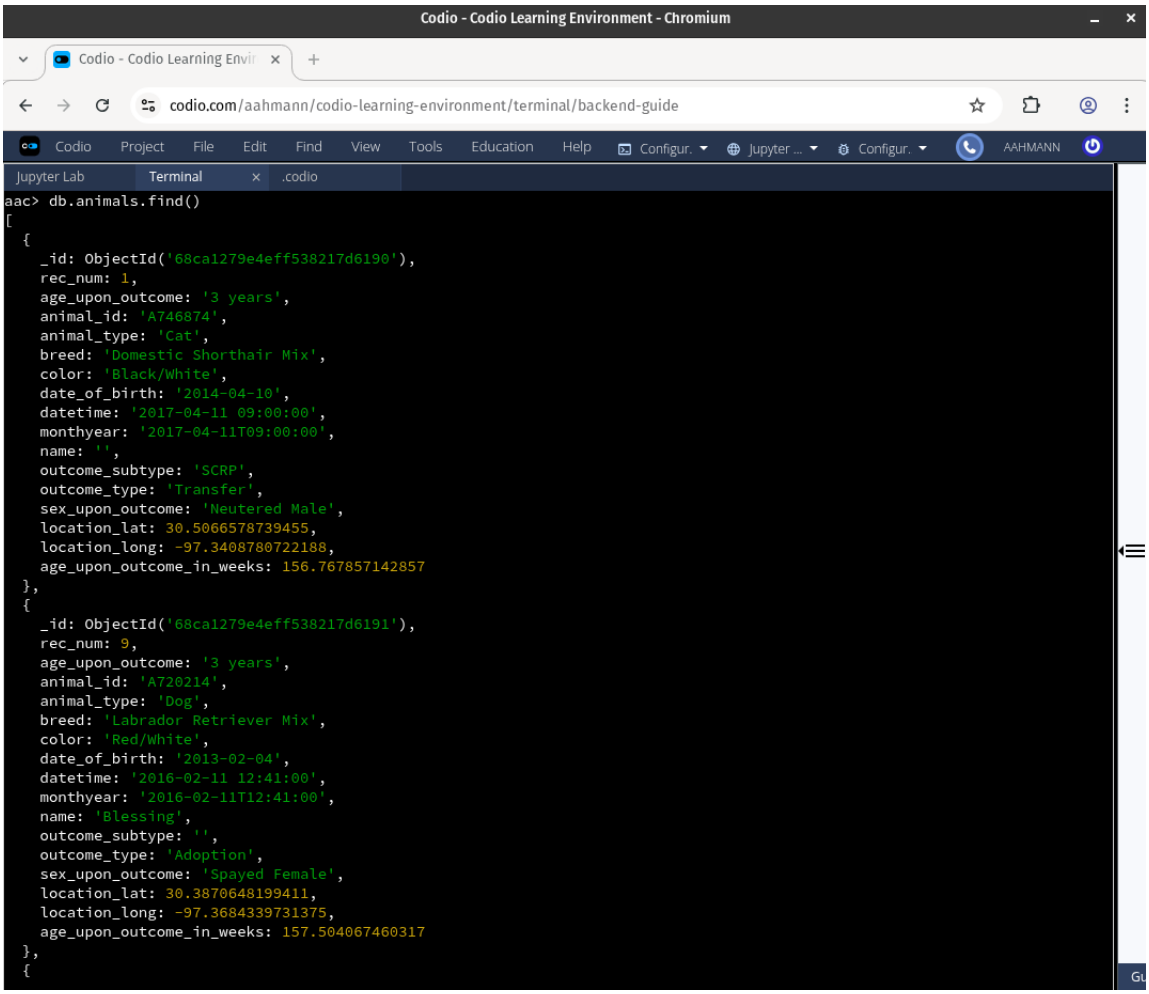

Figure 2: A small sample of documents in the `aac.animals` collection.

B.2 Figures 3, 4 & 5: Creating a single-field index, and performing queries with it

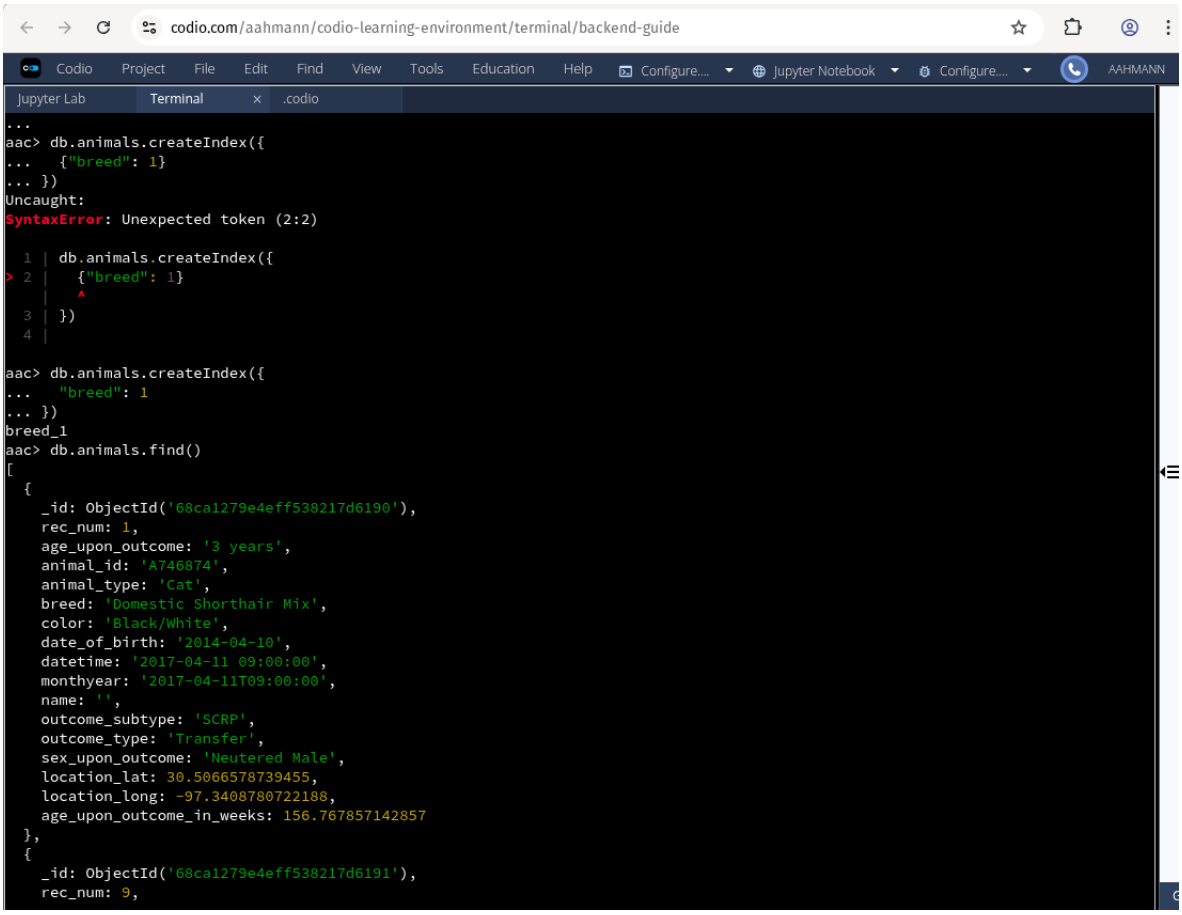

Figure 3: Creating the simple index in accordance with CS-340 (n.d., §1.2).

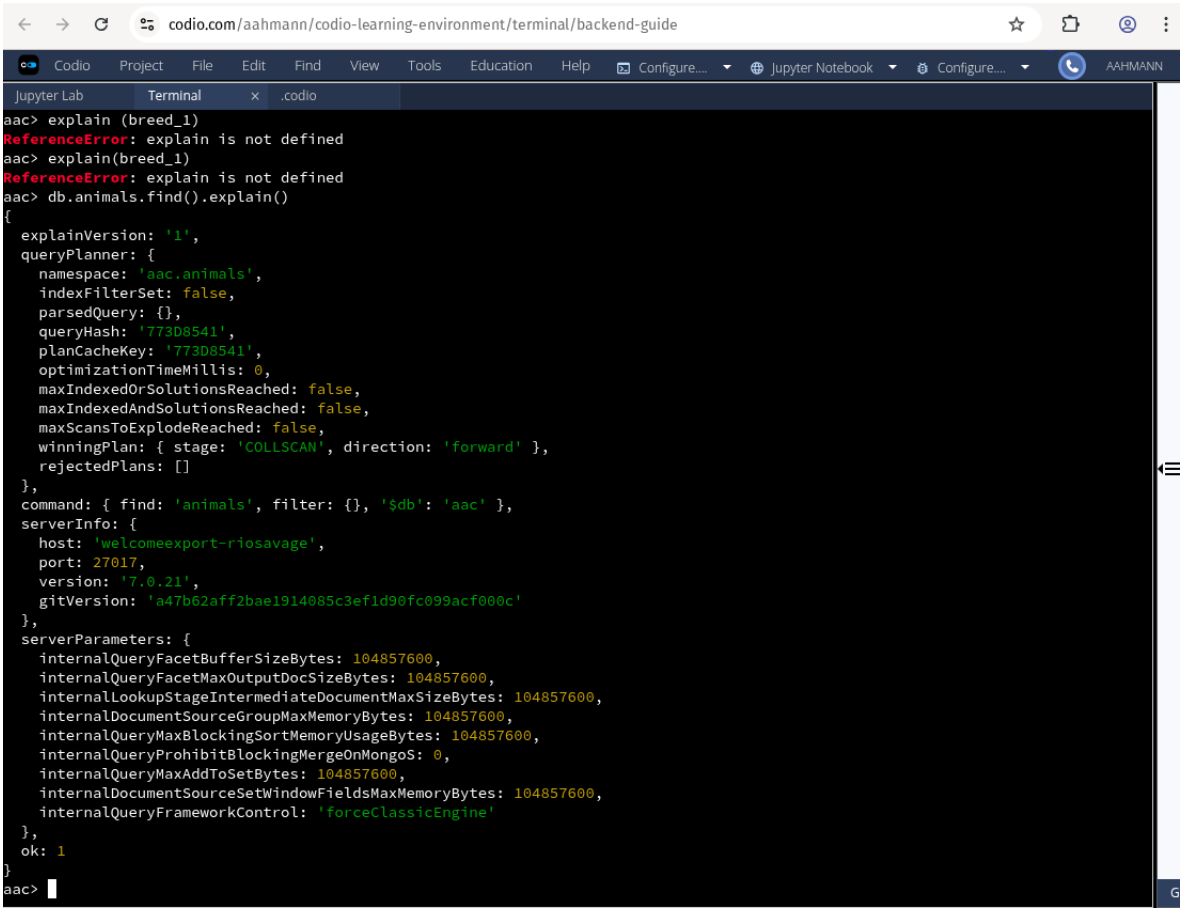

Figure 4: Example of `.explain()` function usage: basic information regarding the `aac.animals` collection.

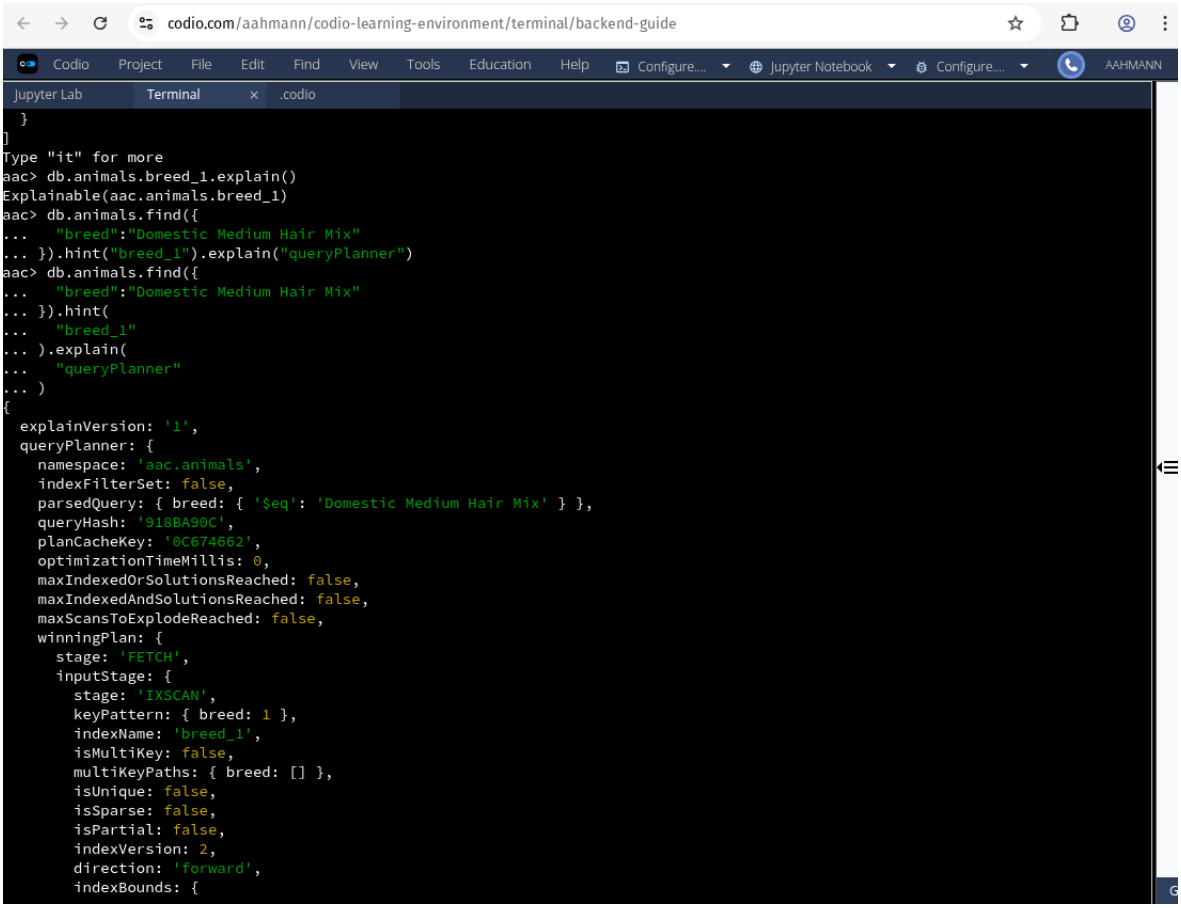

Figure 5: Using the `explain` function on the `breed_1` index, then selecting for data that have the keys `breed` match the value `Domestic Medium Hair Mix`, and finally "explaining" the result's `queryPlanner` attributes.

B.3 Figures 6 & 7: Creating a compound index and using it to optimize `find` queries.

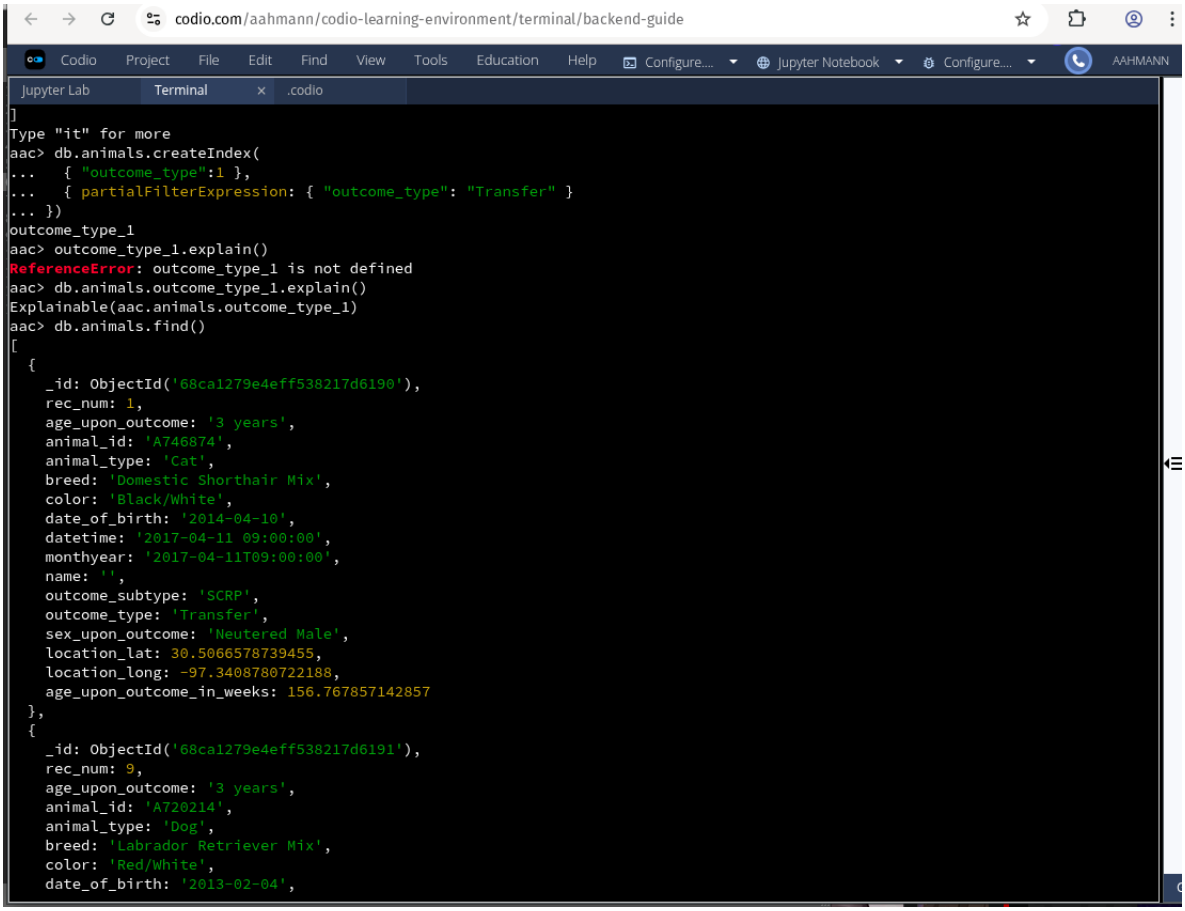

Figure 6: Creating a compound index, confirming its explainability, and sampling documents in the `aac.animals` collection.

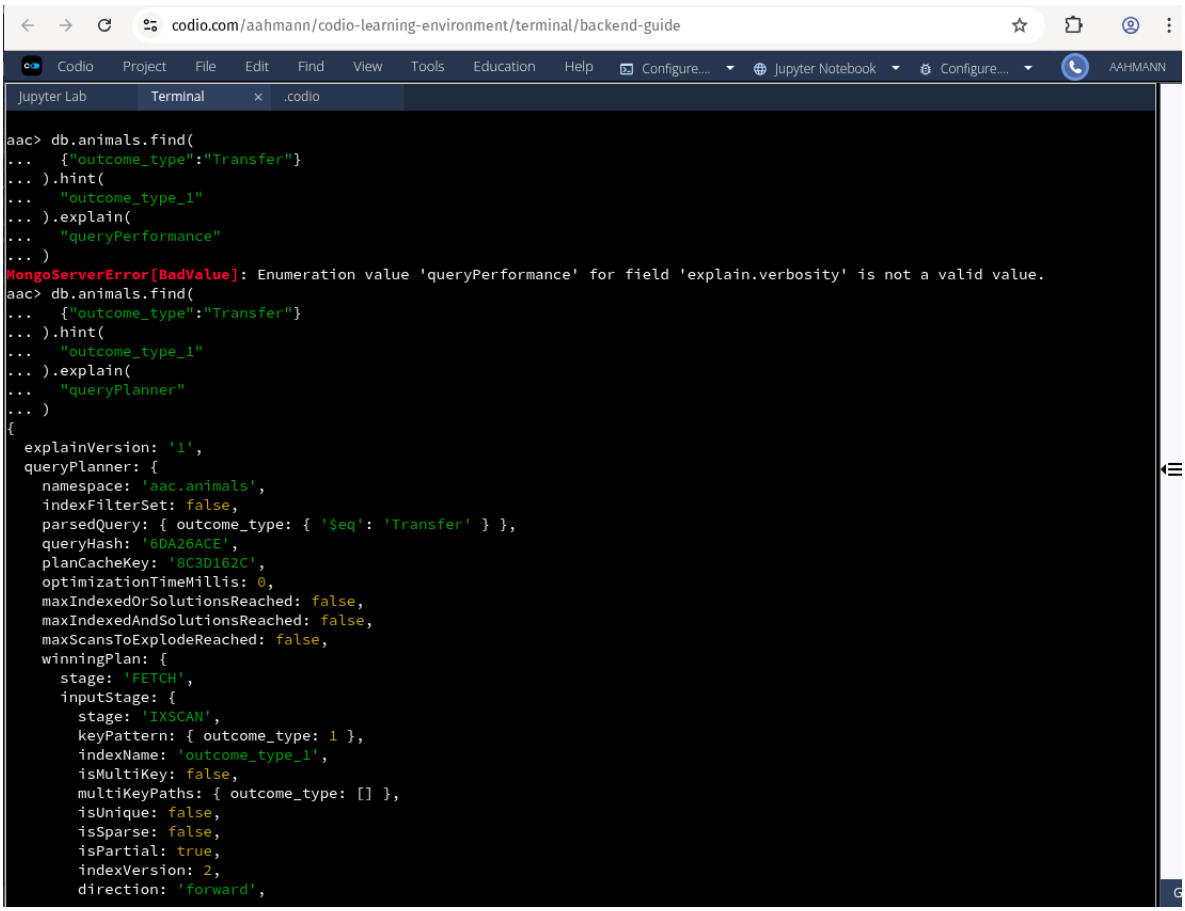

Figure 7: Applying a `hint` parameterized by `outcome_type_1` to query the collection, and explaining its `queryPlanner` results.

B.4 Figures 8 & 9: MongoDB systems administration: adding a new user and confirming its existence

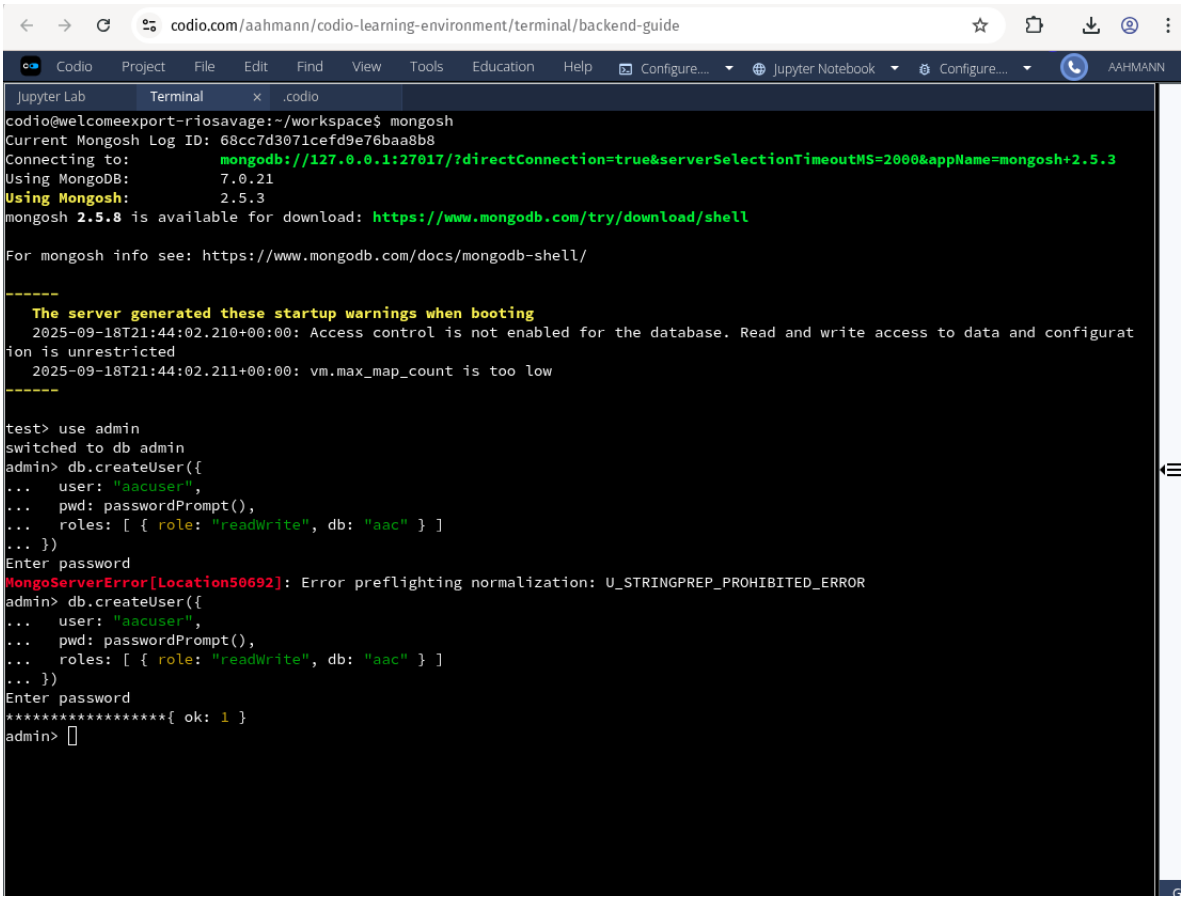

Figure 8: Adding the `aacuser` user account.

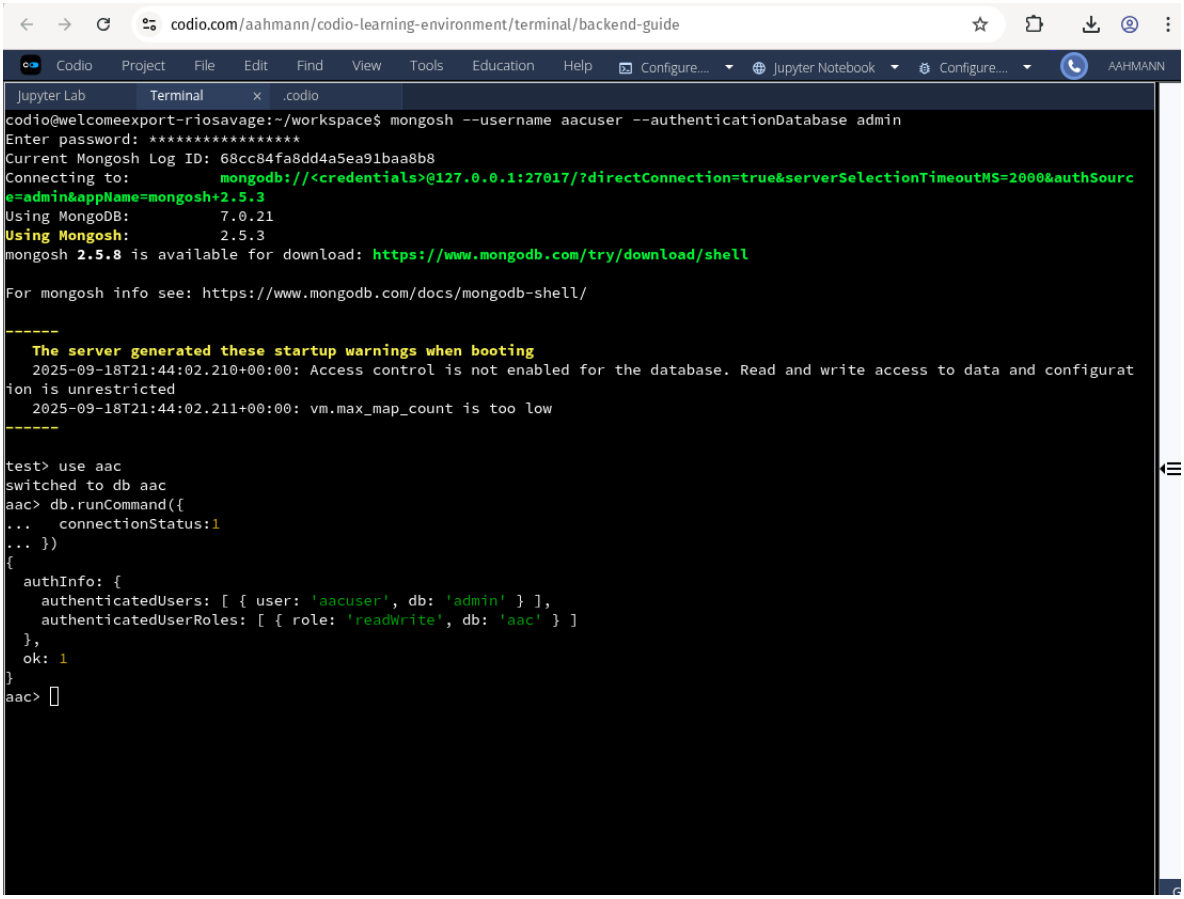

Figure 9: Logging into the MongoDB instance under the `aacuser` and testing connectivity.