CS 628: Full-Stack Development – Web App
City University of Seattle
School of Technology & Computing
Professor Sam Chung

HOS07: MongoDB Atlas

Samantha Hipple
August 12, 2023

PLEASE NOTE

Screenshots in this guide may differ from your environment (e.g., directory paths, version numbers, etc.). When choosing between a stable or most recent release, we advise you install the stable release rather than the best-testing version. Additionally, there may be subtle discrepancies along the steps, please use your best judgment to complete the tutorial. If you are unfamiliar with terminal, command line, and bash scripts, we recommend watching this video prior to moving forward with this guide. Not all steps are fully explained. Lastly, we advise that you avoid copy-pasting code directly from the guide or GitHub repositories. Instead, type out the code yourself to improve familiarity.

More information on this guide can be found under the related module in this repository. Please save a screenshot of the app at the end of each section and save it in the current module folder with the relevant section number.

SECTION CONTENTS

SECTION 1. ACCESSING GITHUB CODESPACES

GitHub Codespaces is an online cloud-based development environment that allows users to easily write, run and debug code. Codespaces is fully integrated with your GitHub repository and provides a seamless experience for developers. In order to access Codespaces, users only need a GitHub account and an active internet connection.

After downloading the current HOS assignment, in the top-right corner of the repo, click on the **`<> Code`** drop-down menu and select **`Create codespace on main`** as shown in the following image. The free and pro GitHub subscriptions include free use of GitHub Codespaces *up to a fixed amount of usage each month*. In order to avoid unexpected charges, please review the billing information.

SECTION 2. SQL VS NOSQL

SQL (Structured Query Language) databases are relational databases with fixed schemas, where data is stored in structured tables with rows and columns. These databases utilize ACID (Atomicity, Consistency, Isolation, Durability) transactions to ensure data integrity. They are optimal for handling structured data and maintaining well-defined relationships, with popular examples being MySQL, PostgreSQL, SQLite, and Oracle.

NoSQL (Not Only SQL) databases, on the other hand, offer flexible and often schema-less storage solutions. Their data can be stored in diverse formats such as documents, key-value pairs, or graphs. Designed for horizontal scalability and high performance, they are particularly suitable for unstructured or semi-structured data. Some renowned NoSQL databases include MongoDB, Cassandra, Redis, and Couchbase.

SECTION 3. INTRODUCTION TO MONGODB

MongoDB is a document-oriented NoSQL database known for storing data in JSON-like documents. Unlike rigid structures, these documents can vary in structure, providing flexibility in how data is represented. Within MongoDB, there are key concepts to understand:

- **Collections** are akin to tables in relational databases and hold related documents.
- **Documents** serve as individual data records and use the BSON (Binary JSON) format.
- **Fields** within these documents can be likened to columns in SQL databases.

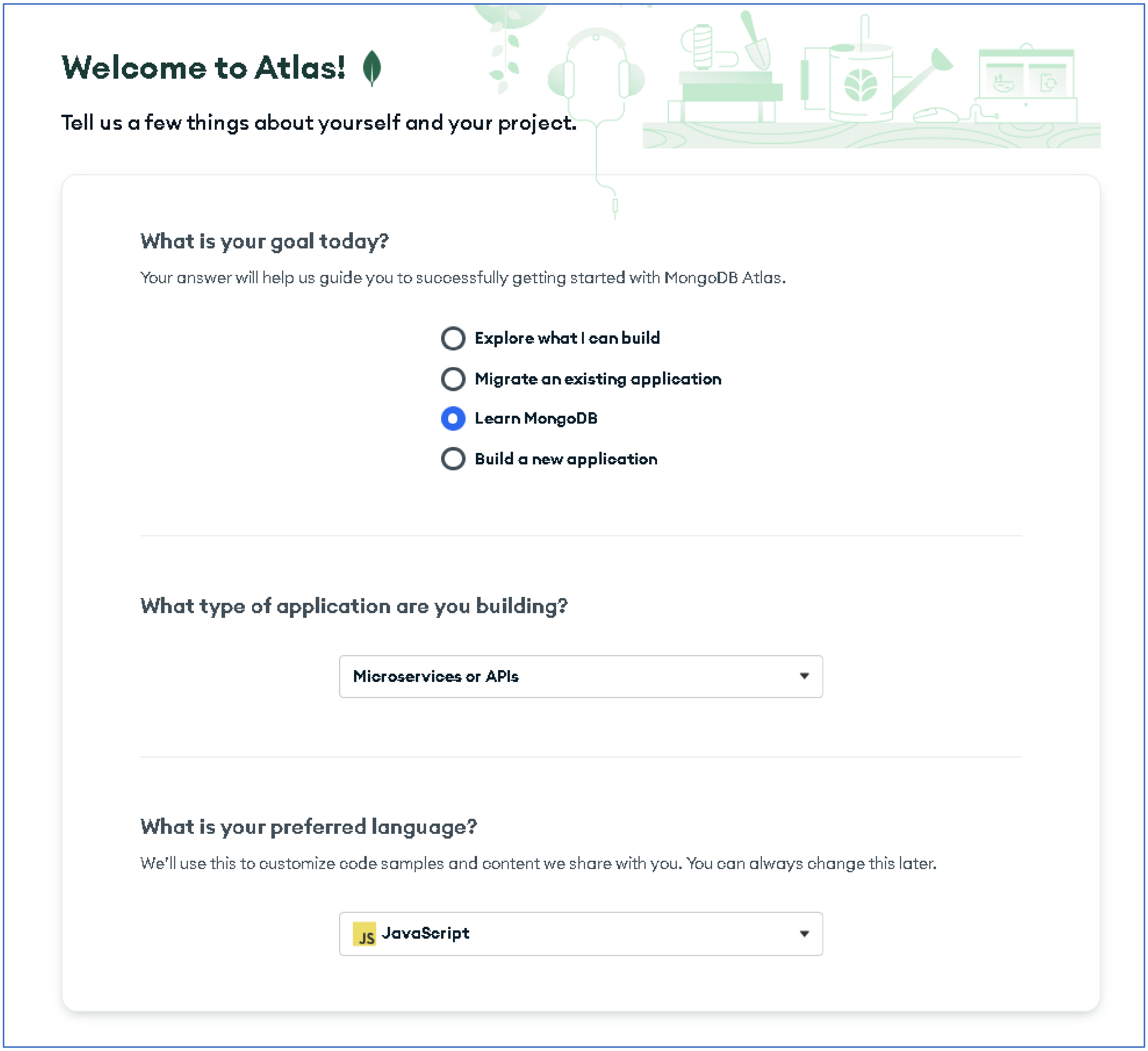
Some clear advantages of using MongoDB include its flexible schema, which adapts to changing data requirements, and its speed in read and write operations. In terms of functionalities, MongoDB supports CRUD (Create, Read, Update, Delete) operations, albeit with its unique syntax. The flexible query language is adept at retrieving specific documents, and users can perform tasks like filtering, sorting, and aggregation.

For advanced data operations, there's the aggregation framework. This involves processes like grouping and transforming data. Additionally, MongoDB allows the creation of indexes on specific fields to boost query performance and speed up read operations; and supports geospatial queries, facilitating location-based data queries. Given these capabilities, MongoDB is an ideal choice for various applications, such as websites, content management systems, cataloging, and real-time analytics.

SECTION 4. SIGNING UP FOR MONGODB ATLAS

1. Open this link: https://www.mongodb.com/atlas/database and click on **Try Free**.
2. Either complete the registration form or sign up with a Gmail account.

3. Once registered, sign in and complete the welcome form as demonstrated below:



Welcome to Atlas!

Tell us a few things about yourself and your project.

What is your goal today?

Your answer will help us guide you to successfully getting started with MongoDB Atlas.

○ Explore what I can build
○ Migrate an existing application
● Learn MongoDB
○ Build a new application

What type of application are you building?

Microservices or APIs ▾

What is your preferred language?

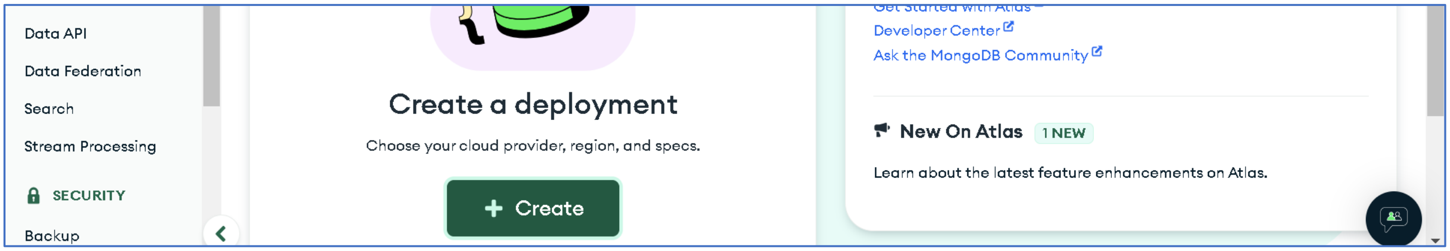
We'll use this to customize code samples and content we share with you. You can always change this later.

JS JavaScript ▾

4. Press the **Finish** button below the form to submit.

SECTION 5. SETTING UP A MONGODB DATABASE

1. Create a new deployment:

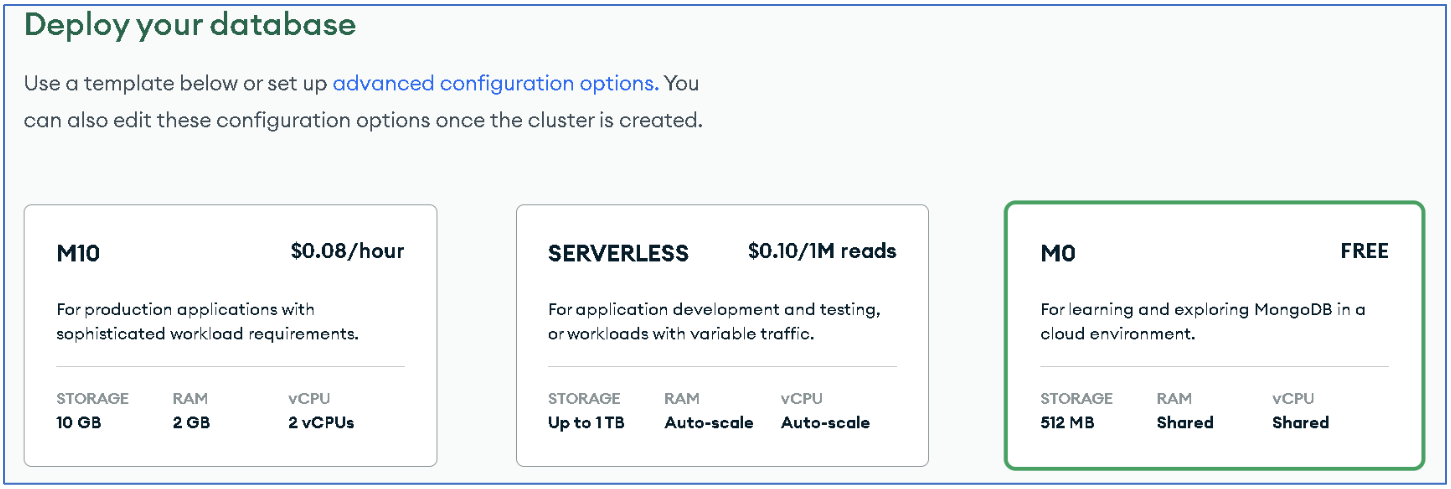


Create a deployment

Choose your cloud provider, region, and specs.

Learn about the latest feature enhancements on Atlas.

2. Select the **M0 FREE** template:



Deploy your database

Use a template below or set up advanced configuration options. You can also edit these configuration options once the cluster is created.

M10 $0.08/hour

For production applications with sophisticated workload requirements.

STORAGE	RAM	vCPU
10 GB	2 GB	2 vCPUs

SERVERLESS $0.10/1M reads

For application development and testing, or workloads with variable traffic.

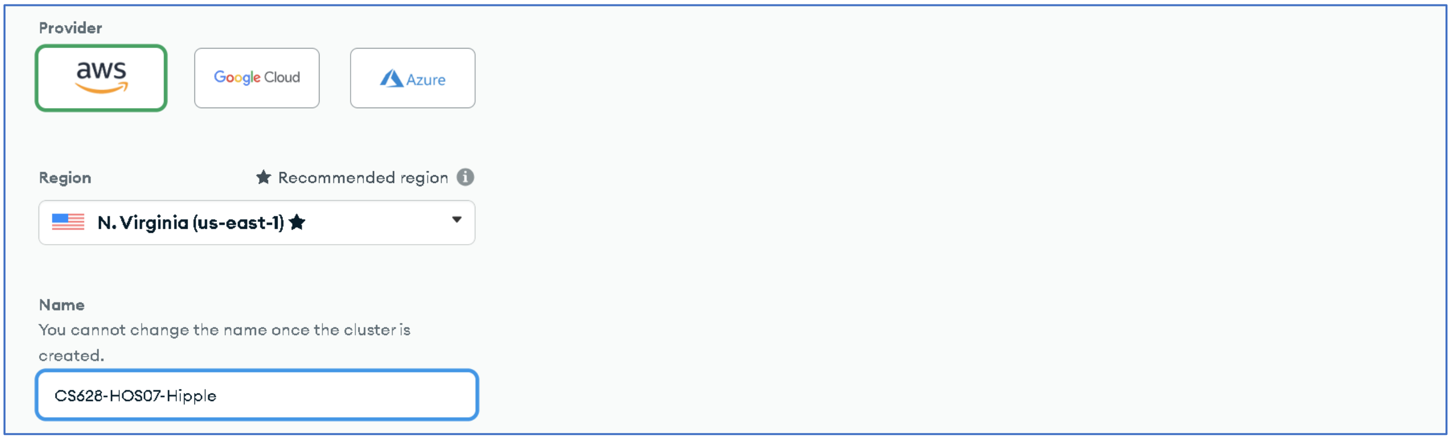
STORAGE	RAM	vCPU
Up to 1 TB	Auto-scale	Auto-scale

M0 FREE

For learning and exploring MongoDB in a cloud environment.

STORAGE	RAM	vCPU
512 MB	Shared	Shared

3. Set **AWS** as the provider and provide a name for the new cluster:



Provider

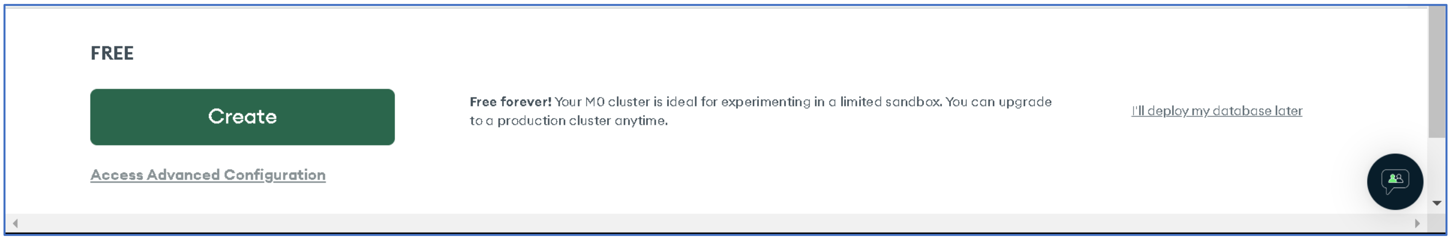
Region ★ Recommended region

N. Virginia (us-east-1) ★

Name
You cannot change the name once the cluster is created.

CS628-HOS07-Hipple

4. Click **Create** once the above steps are completed:

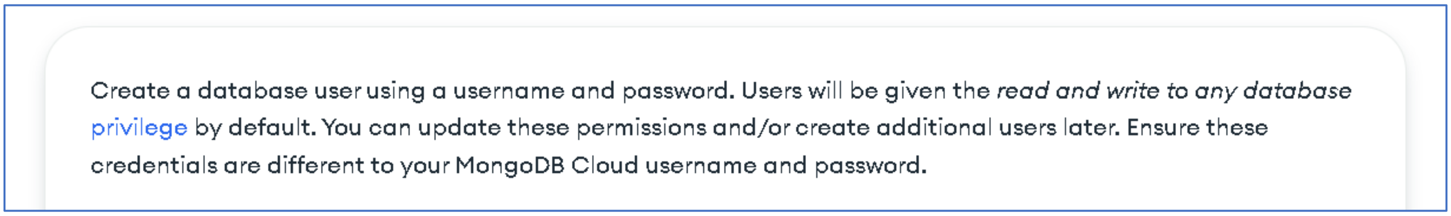


FREE

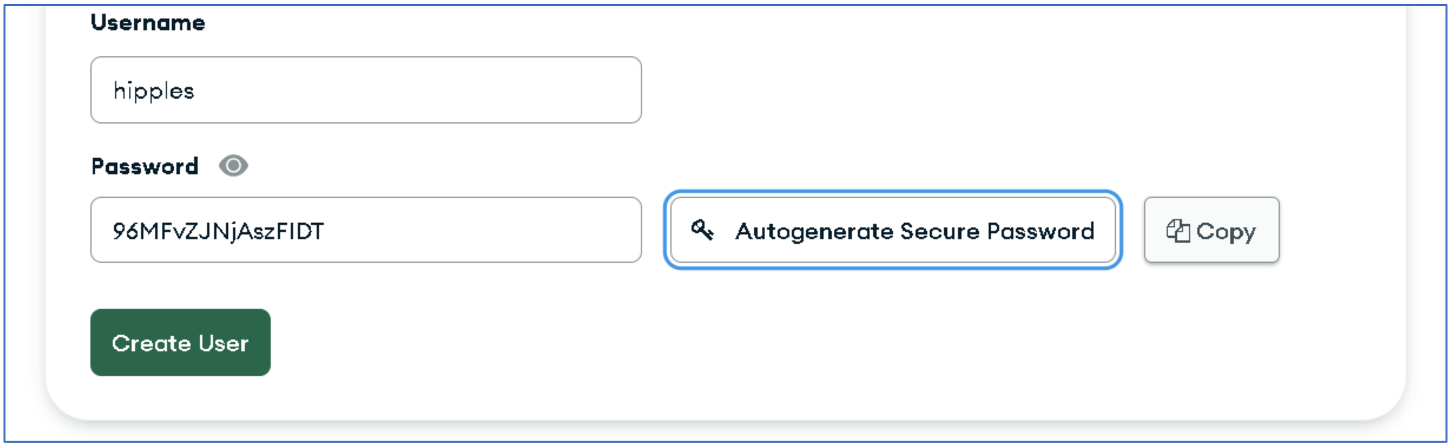
Free forever! Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

Once submitted, you will be redirected to a Security QuickStart screen.

5. Create a username and password for the MongoDB connection.



Create a database user using a username and password. Users will be given the *read and write to any database* privilege by default. You can update these permissions and/or create additional users later. Ensure these credentials are different to your MongoDB Cloud username and password.

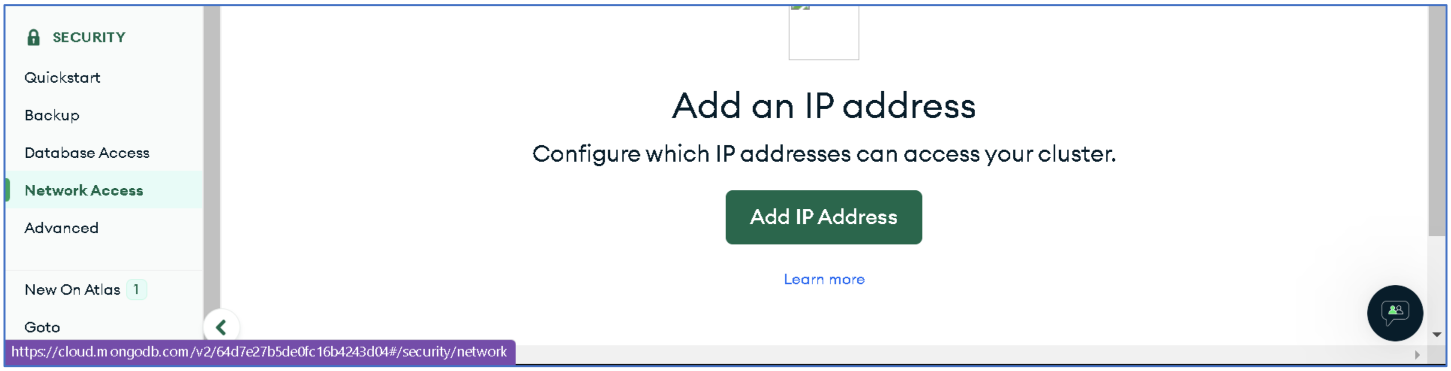

Username: hipples

Password: 96MFvZJNjAszFIDT

6. Then press the **Create User** button.

Remember the credentials for the new user account. They are needed to create a connection string further on in the tutorial. Next, we need to enable access to Atlas from anywhere. Ideally, we would only allow connections to MongoDB Atlas from trusted IP addresses. For now, we will use wild cards that allow universal access.

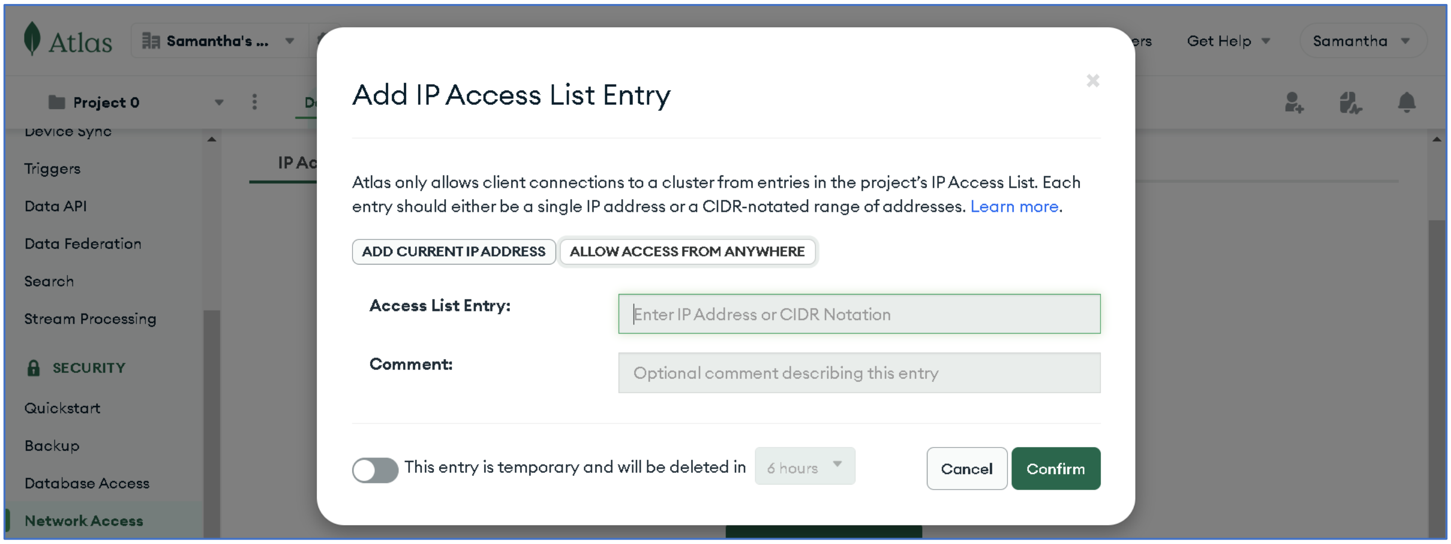
7. Select the **Network Access** tab in the side menu and click the **Add IP Address** button:


Add an IP address

Configure which IP addresses can access your cluster.

https://cloud.m ongodb.com/v2/64d7e27b5de0fc16b4243d04#/security/network

8. Select **ALLOW ACCESS FROM ANYWHERE**, then press the **Confirm** button.


Add IP Access List Entry

Atlas only allows client connections to a cluster from entries in the project's IP Access List. Each entry should either be a single IP address or a CIDR-notated range of addresses. Learn more.

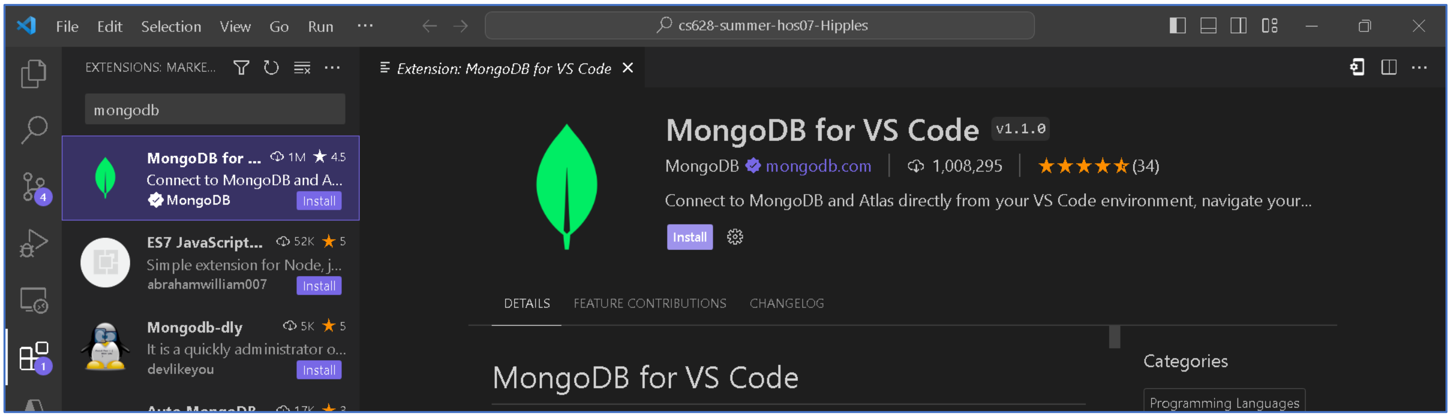
Access List Entry: Enter IP Address or CIDR Notation

Comment: Optional comment describing this entry

This entry is temporary and will be deleted in 6 hours

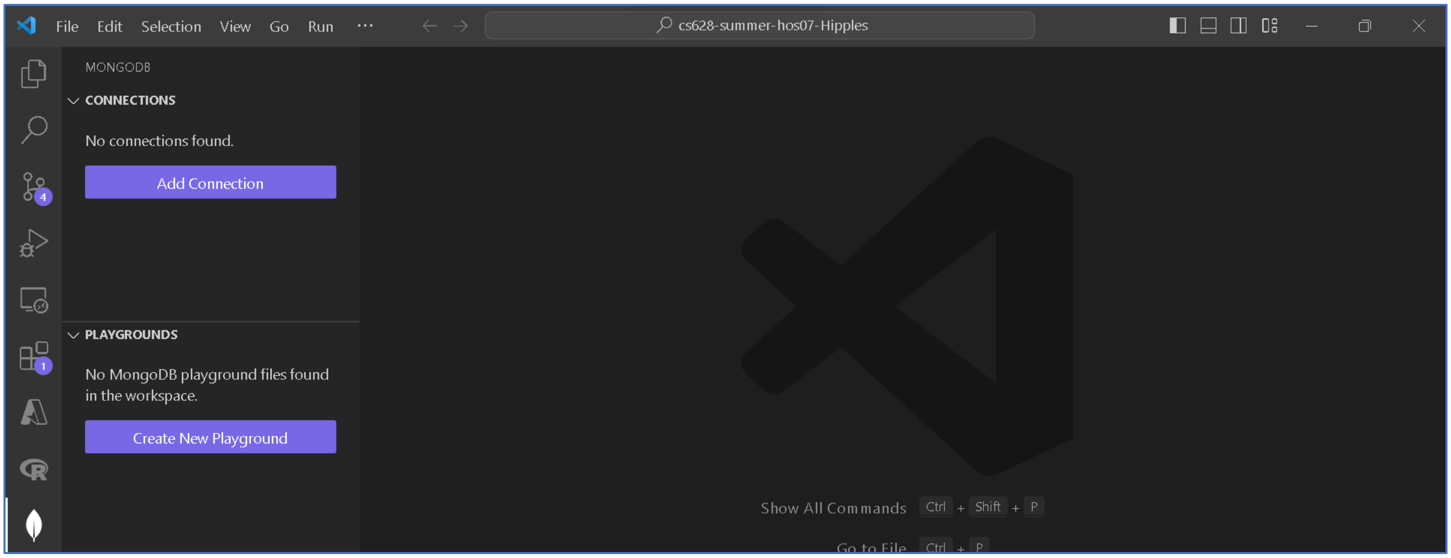
SECTION 6. SETTING UP A MONGOBD ENVIRONMENT

1. Install the **`MongoDB for VS Code`** extension in your development environment:


File Edit Selection View Go Run ···
cs628-summer-hos07-Hipples
EXTENSIONS: MARKE...
mongodb
MongoDB for ... 1M ★ 4.5
Connect to MongoDB and A...
MongoDB Install
ES7 JavaScript... 52K ★ 5
Simple extension for Node, j...
abrahamwilliam007 Install
Mongodb-dly 5K ★ 5
It is a quickly administrator o...
devlikeyou Install
Extension: MongoDB for VS Code
MongoDB for VS Code v1.1.0
MongoDB mongodb.com 1,008,295 ★★★★★(34)
Install
DETAILS FEATURE CONTRIBUTIONS CHANGELOG
MongoDB for VS Code
Categories
Programming Languages

Connect to MongoDB and Atlas directly from your VS Code environment, navigate your...

2. Navigate to the extension in the side menu, then click on **`Add Connection`**:

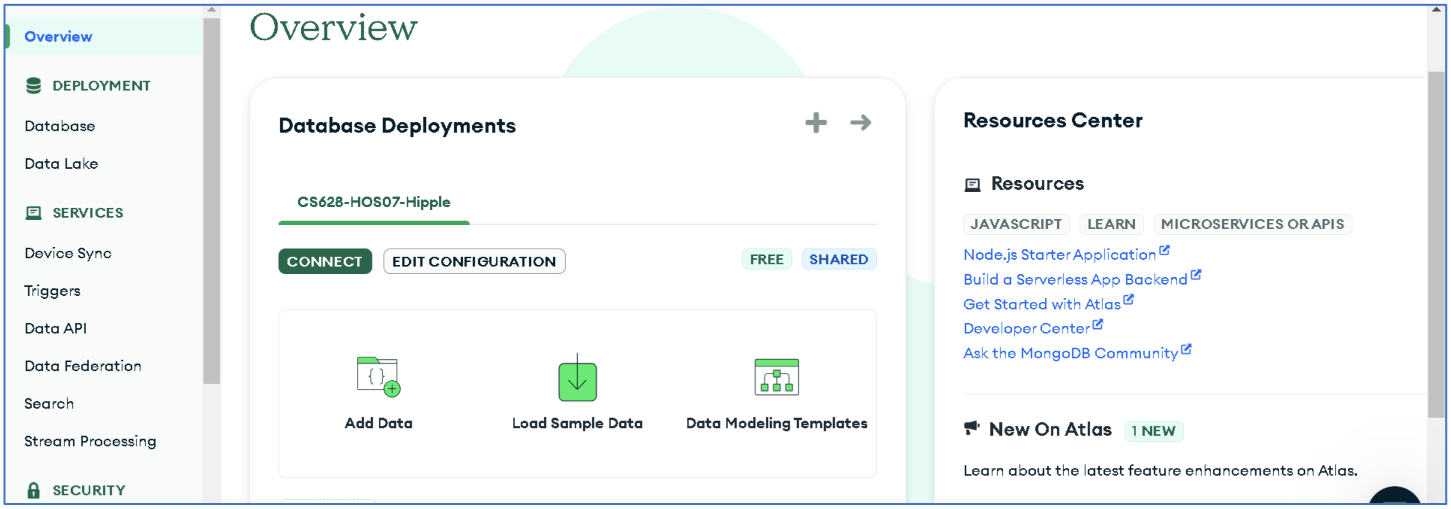

File Edit Selection View Go Run ···
cs628-summer-hos07-Hipples
MONGODB
CONNECTIONS
Add Connection
PLAYGROUNDS
Create New Playground
Show All Commands Ctrl + Shift + P
Go to File Ctrl + P

No connections found.

No MongoDB playground files found in the workspace.

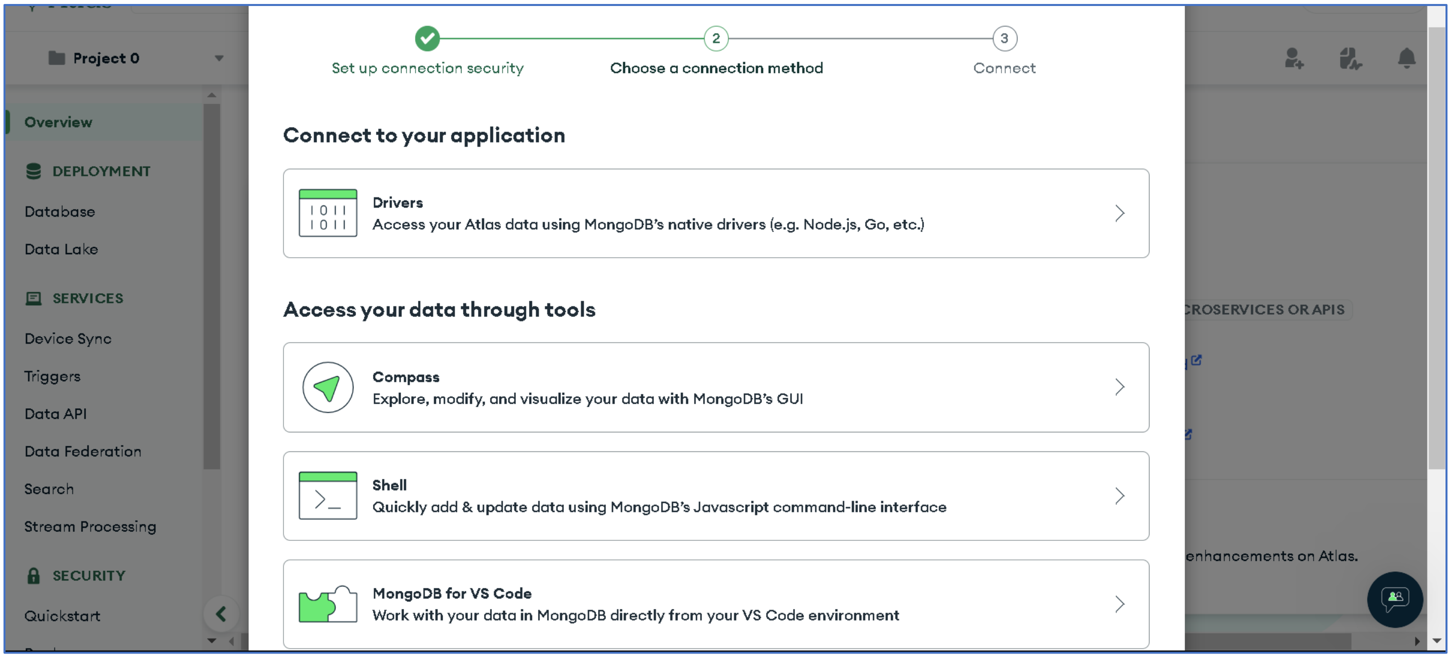
Next, we need to retrieve our connection string from the **`Overview`** section in our Atlas account.

3. Click on **`CONNECT`** for the deployment we just created:



Learn about the latest feature enhancements on Atlas.

4. Under **Access you data through tools**, select **MongoDB for VS Code**:



Connect to your application

Drivers
Access your Atlas data using MongoDB's native drivers (e.g. Node.js, Go, etc.)

Access your data through tools

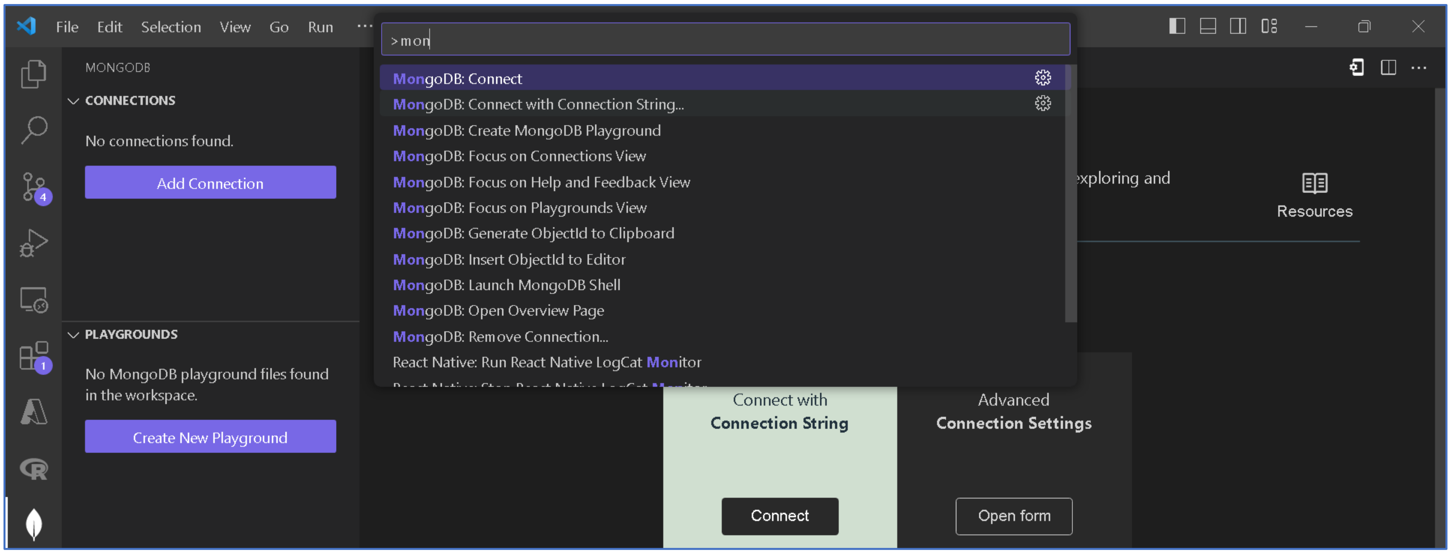
Compass
Explore, modify, and visualize your data with MongoDB's GUI

Shell
Quickly add & update data using MongoDB's Javascript command-line interface

MongoDB for VS Code
Work with your data in MongoDB directly from your VS Code environment

Follow the directions on the next screen, which are outlined in more detail in the following steps. We already completed step one by installing the extension.

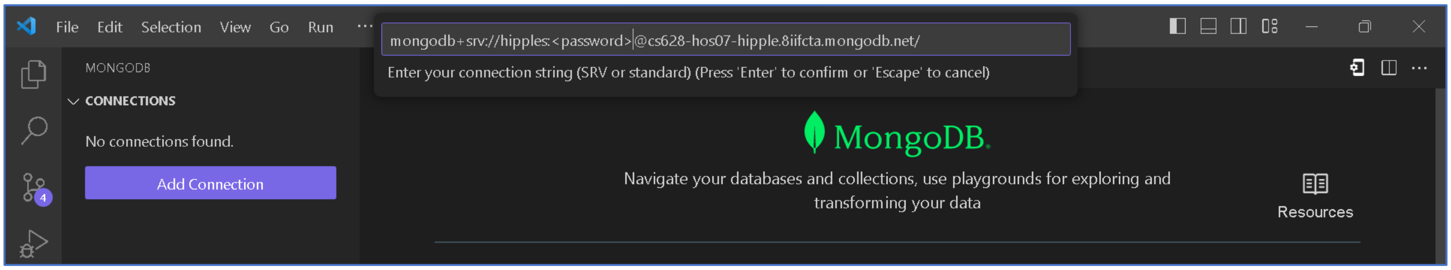
5. Return to your VS Code window and navigate to the command palette.
6. Search for **MongoDB: Connect**, then select **Connect with Connection String**:


File
Edit
Selection
View
Go
Run
...
MONGODB
CONNECTIONS
Add Connection
PLAYGROUNDS
Create New Playground
>mon
MongoDB: Connect
MongoDB: Connect with Connection String...
MongoDB: Create MongoDB Playground
MongoDB: Focus on Connections View
MongoDB: Focus on Help and Feedback View
MongoDB: Focus on Playgrounds View
MongoDB: Generate ObjectId to Clipboard
MongoDB: Insert ObjectId to Editor
MongoDB: Launch MongoDB Shell
MongoDB: Open Overview Page
MongoDB: Remove Connection...
React Native: Run React Native LogCat Monitor
xploring and
Resources
Connect with Connection String
Connect
Advanced Connection Settings
Open form

No connections found.

No MongoDB playground files found in the workspace.

7. Copy the connection string prompt from Atlas to paste into the command palette.
8. Replace **<password>** with the password created for the new user we set up previously:



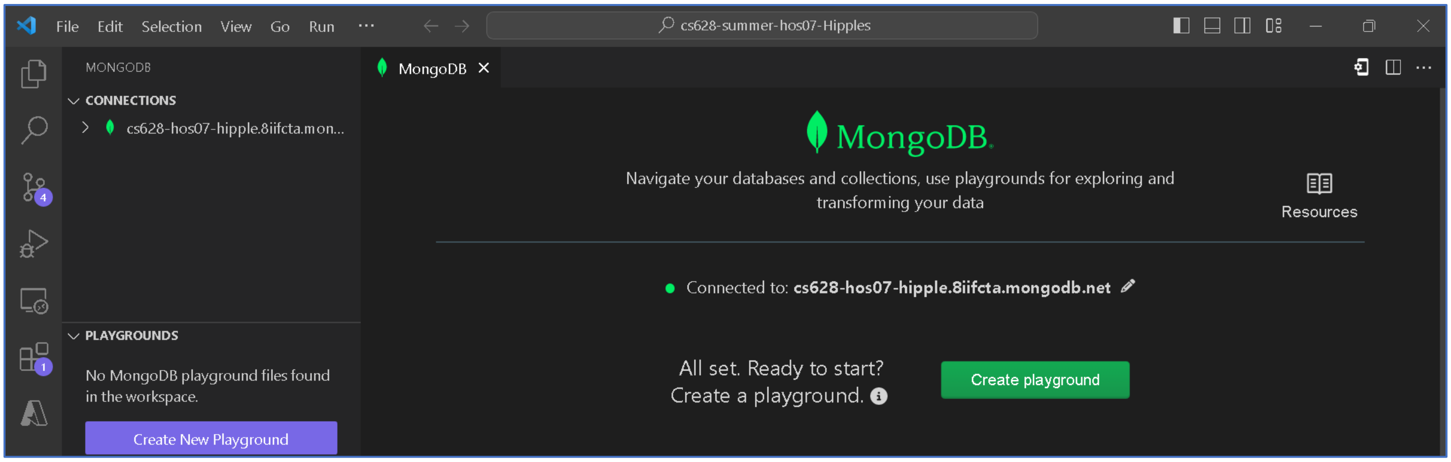
```
mongodb+srv://hipples:<password>@cs628-hos07-hipple.8iifcta.mongodb.net/
```

Enter your connection string (SRV or standard) (Press 'Enter' to confirm or 'Escape' to cancel)

No connections found.

Navigate your databases and collections, use playgrounds for exploring and transforming your data

9. Press **Enter** to establish the connection:


File Edit Selection View Go Run ···
cs628-summer-hos07-Hipples
MONGODB
CONNECTIONS
cs628-hos07-hipple.8iifcta.mon...
PLAYGROUNDS
Create New Playground
MongoDB
Resources
Create playground

No MongoDB playground files found in the workspace.

MongoDB

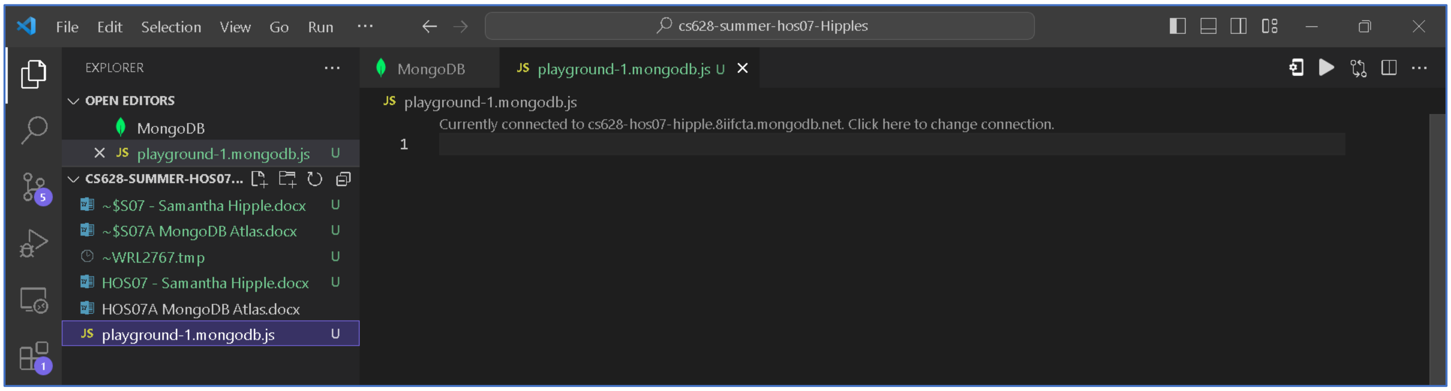
Navigate your databases and collections, use playgrounds for exploring and transforming your data

● Connected to: **cs628-hos07-hipple.8iifcta.mongodb.net**

All set. Ready to start?
Create a playground.

NOTE: *If you user password contains special characters such as* **: / # [ ]** *or* **@**, *please follow the directions in [this link] to encode them for your connection string.*

10. Click the **`Create playground`** button, then remove the generated script for a blank slate.
11. Save the blank file to proper directory for this module (**`HOS07`**):


File Edit Selection View Go Run ···
cs628-summer-hos07-Hipples
EXPLORER
OPEN EDITORS
MongoDB
playground-1.mongodb.js U
CS628-SUMMER-HOS07...
~$S07 - Samantha Hipple.docx U
~$S07A MongoDB Atlas.docx U
~WRL2767.tmp U
HOS07 - Samantha Hipple.docx U
HOS07A MongoDB Atlas.docx
playground-1.mongodb.js U
MongoDB
playground-1.mongodb.js U
playground-1.mongodb.js

Currently connected to cs628-hos07-hipple.8iifcta.mongodb.net. Click here to change connection.

SECTION 7. LEARNING MONGODB STEP-BY-STEP

Finally, we are going to learn MongoDB commands by executing them one-by-one in the playground we just created. MongoDB playgrounds are interactive JavaScript (JS) environments that are designed for prototyping queries, aggregations, and MongoDB commands. These playgrounds provide useful syntax highlighting to assist in development. In MongoDB for VS Code, playgrounds are recognized by files bearing the **`.mongodb.js`** extension.

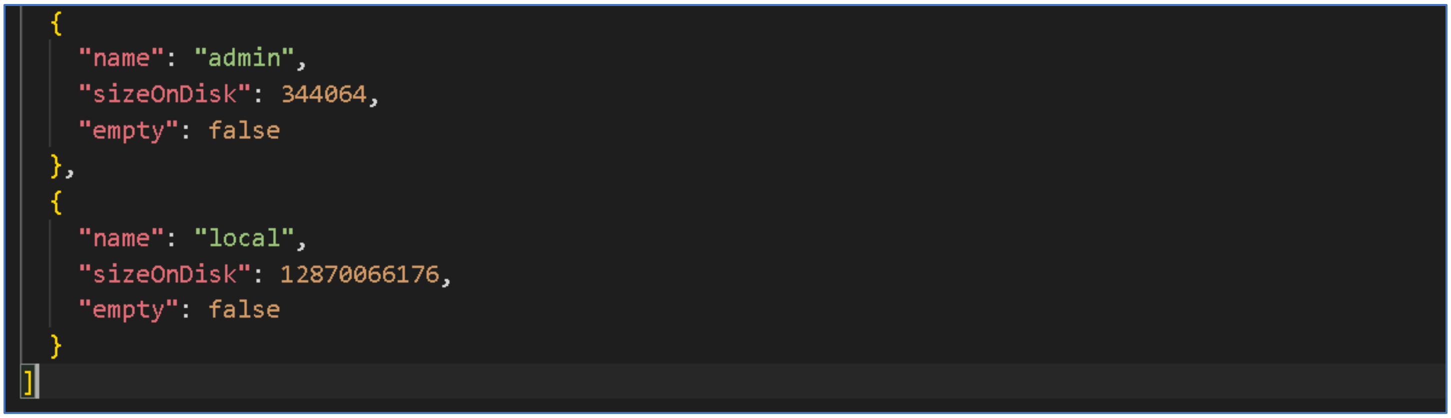
NOTE: *In order to run the commands in the following steps, press the play button found in the top right corner of the playground file. Capture screenshots of the playground results as you go for submission.*

1. Display a list of available databases from the current MongoDB server instance:

```
show dbs
```

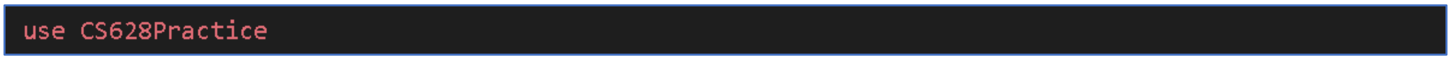
Playground output:

```
[
```



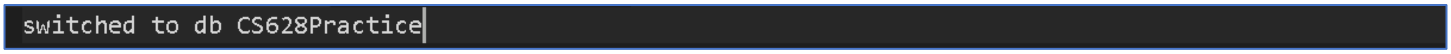
```
  {
    "name": "admin",
    "sizeOnDisk": 344064,
    "empty": false
  },
  {
    "name": "local",
    "sizeOnDisk": 12870066176,
    "empty": false
  }
]
```

2. Switch to or create a specific database using the command below:



```
use CS628Practice
```

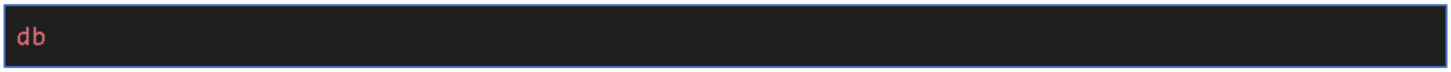
Playground output:



```
switched to db CS628Practice
```

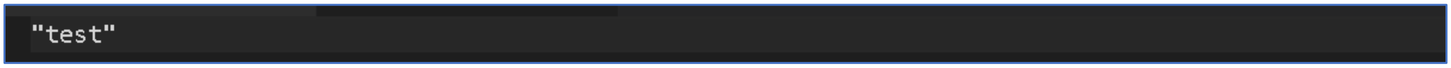
If the database does not exist, MongoDB will create it.

3. Display the name of the currently selected database:



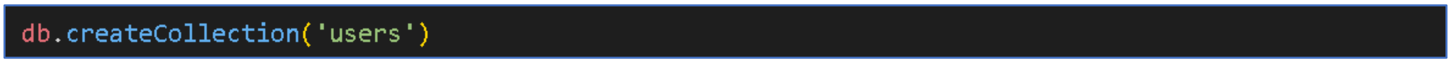
```
db
```

Playground output:



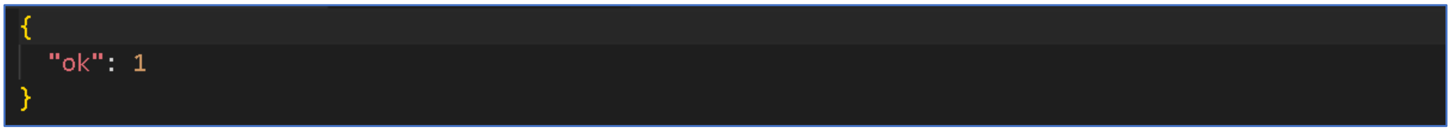
```
"test"
```

4. Create a new collection named **users** within the current database:



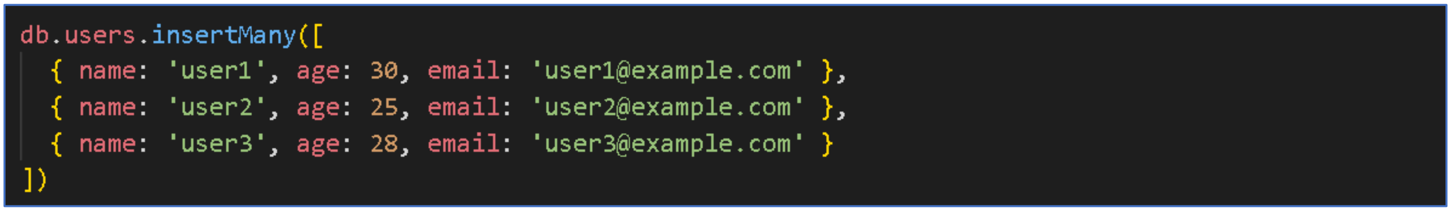
```
db.createCollection('users')
```

Playground output:



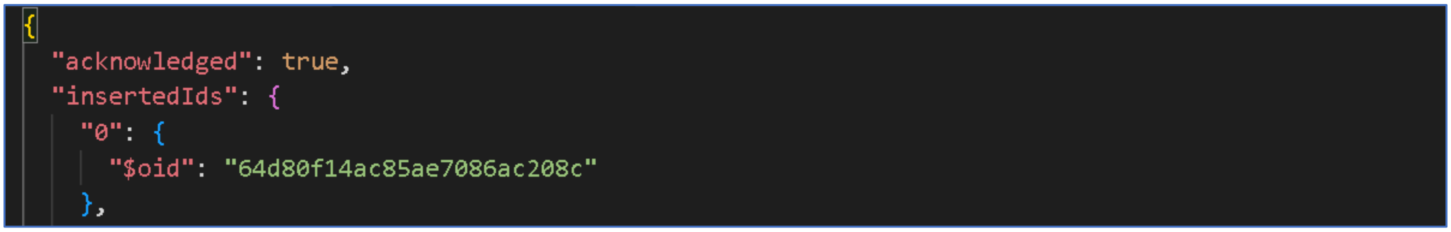
```
{
  "ok": 1
}
```

5. Insert multiple user profiles into the **users** collection:

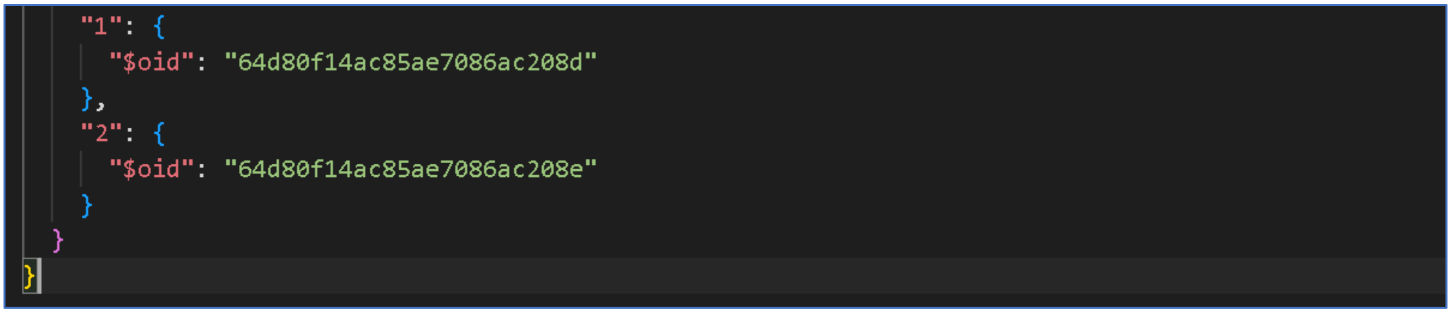


```
db.users.insertMany([
  { name: 'user1', age: 30, email: 'user1@example.com' },
  { name: 'user2', age: 25, email: 'user2@example.com' },
  { name: 'user3', age: 28, email: 'user3@example.com' }
])
```

Playground output:



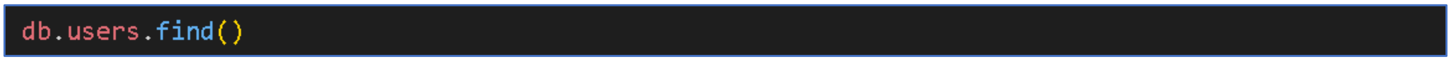
```
{
  "acknowledged": true,
  "insertedIds": {
    "0": {
      "$oid": "64d80f14ac85ae7086ac208c"
    },
```



```
    "1": {
      "$oid": "64d80f14ac85ae7086ac208d"
    },
    "2": {
      "$oid": "64d80f14ac85ae7086ac208e"
    }
  }
}
```

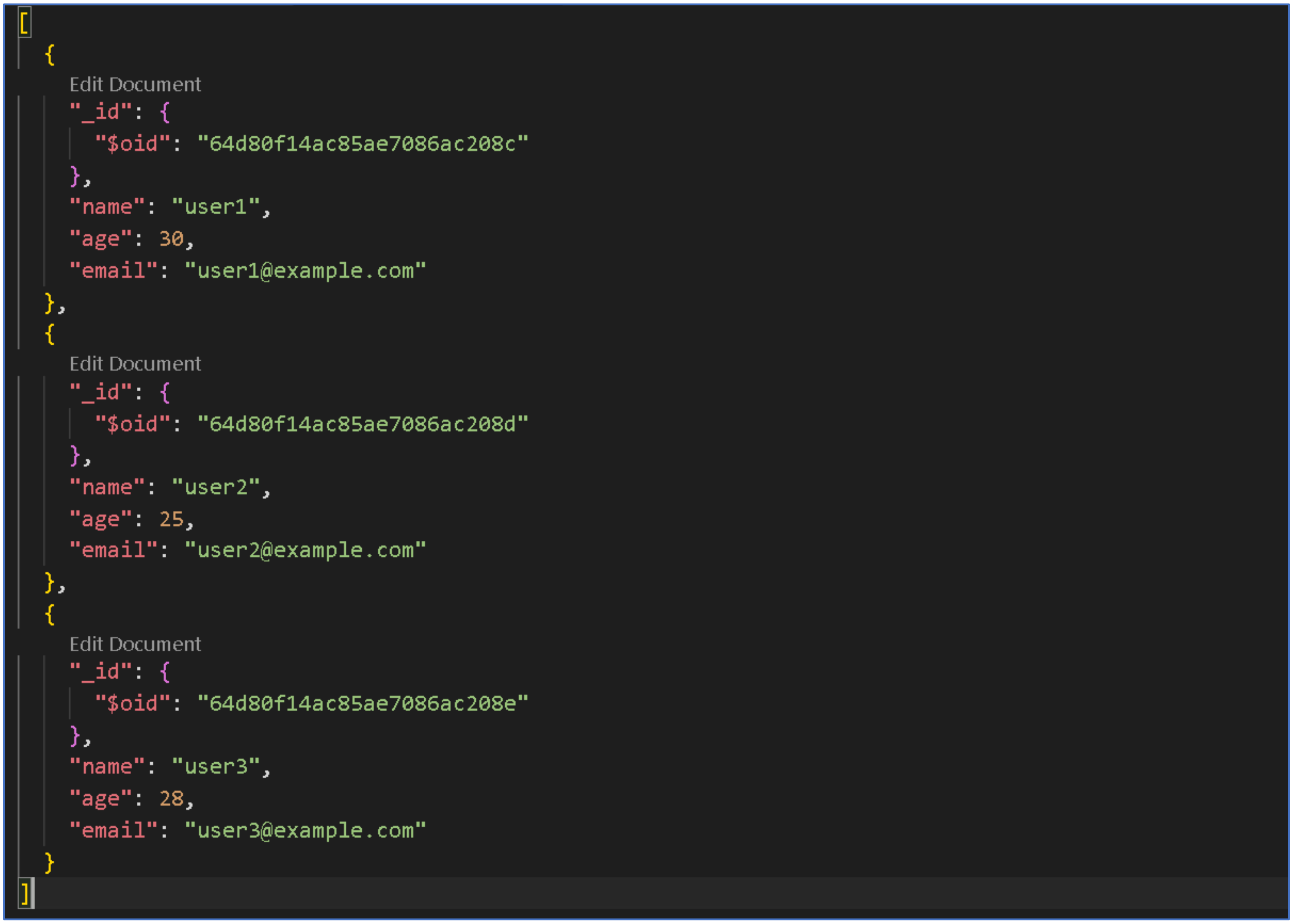
Each user document contains three fields: **name**, **age**, and **email**.

6. Display all documents from the **users** collection:



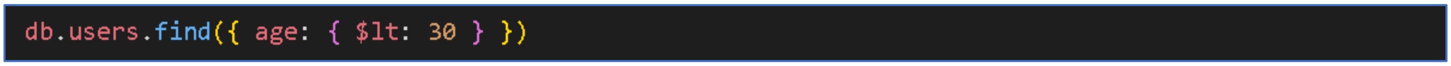
```
db.users.find()
```

Playground output:



```
[
  {
    Edit Document
    "_id": {
      "$oid": "64d80f14ac85ae7086ac208c"
    },
    "name": "user1",
    "age": 30,
    "email": "user1@example.com"
  },
  {
    Edit Document
    "_id": {
      "$oid": "64d80f14ac85ae7086ac208d"
    },
    "name": "user2",
    "age": 25,
    "email": "user2@example.com"
  },
  {
    Edit Document
    "_id": {
      "$oid": "64d80f14ac85ae7086ac208e"
    },
    "name": "user3",
    "age": 28,
    "email": "user3@example.com"
  }
]
```

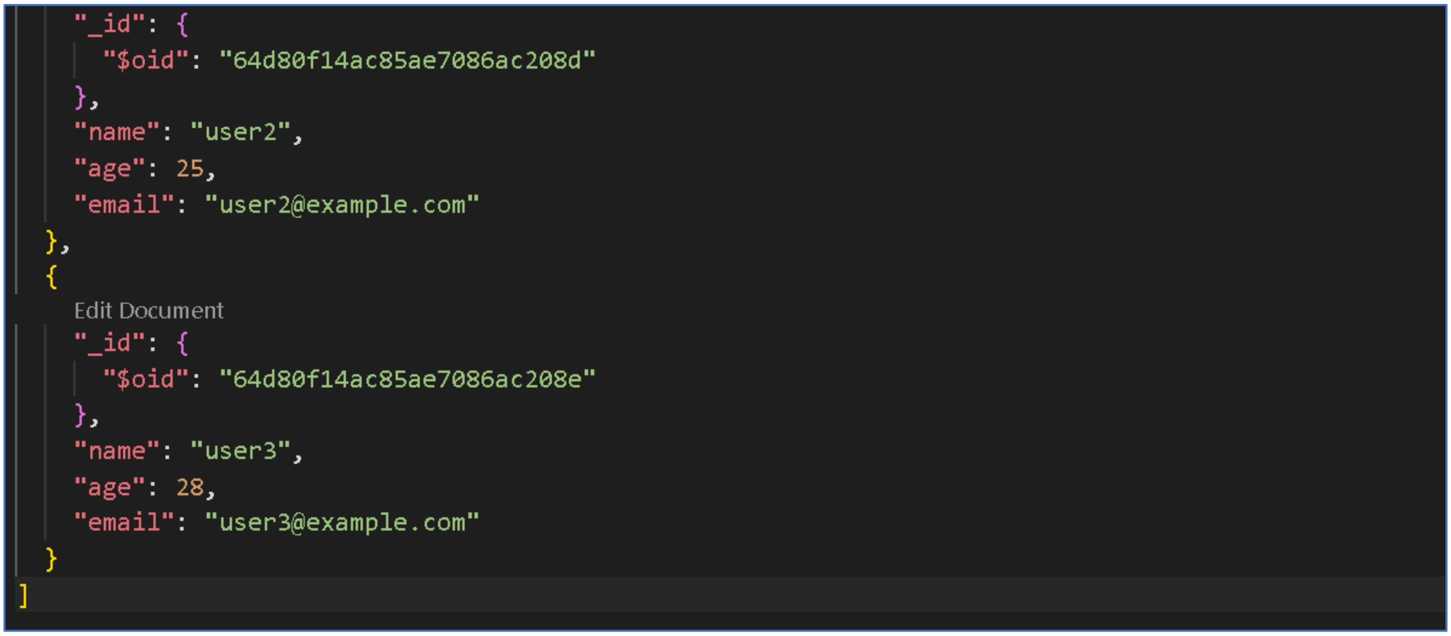
7. Display only user documents whose **age** field contains a value of less than **30**:



```
db.users.find({ age: { $lt: 30 } })
```

Playground output:

```
[
  {
    Edit Document
```

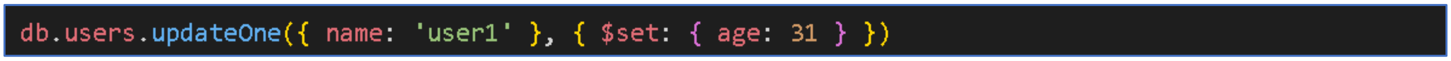


```
    "_id": {
      "$oid": "64d80f14ac85ae7086ac208d"
    },
    "name": "user2",
    "age": 25,
    "email": "user2@example.com"
  },
  {
    Edit Document
    "_id": {
      "$oid": "64d80f14ac85ae7086ac208e"
    },
    "name": "user3",
    "age": 28,
    "email": "user3@example.com"
  }
]
```

Other operators similar to **$lt** (less than) in MongoDB can be found in the chart below.

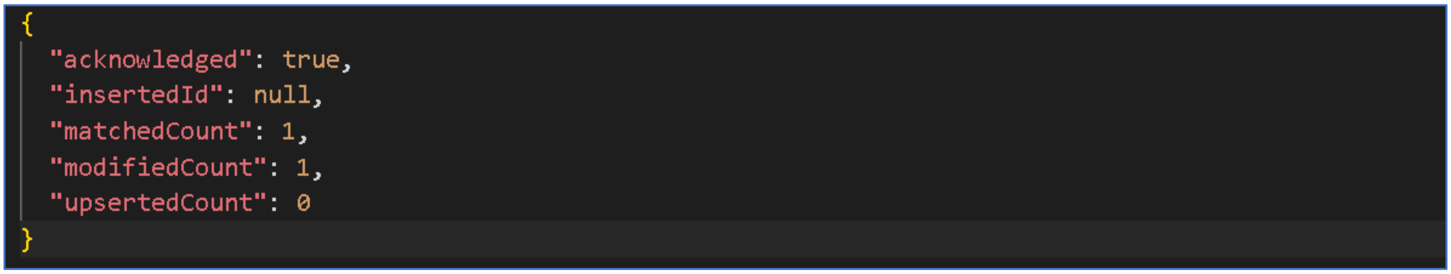
operator	**matches**	**criteria**
$gt	values	that are greater than a specified value
$lte	values	that are less than or equal to a specified value
$gte	values	that are greater than or equal to a specified value
$eq	values	that are equal to a specified
$ne	values	that are not equal to a specified value
$in	values	that exist in a specified array
$nin	values	that do not exist in a specified array
$exists	documents	that have the specified field
$type	documents	where the value of a field has a specific data type

8. Update a single user document in the **users** collection:



```
db.users.updateOne({ name: 'user1' }, { $set: { age: 31 } })
```

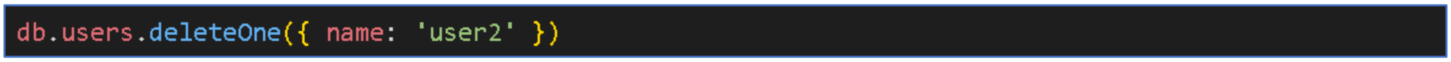
Playground output:



```
{
  "acknowledged": true,
  "insertedId": null,
  "matchedCount": 1,
  "modifiedCount": 1,
  "upsertedCount": 0
}
```

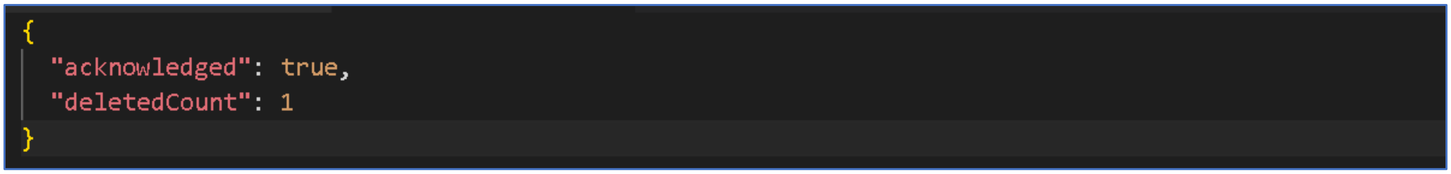
The command above finds the document with the name **user1** and sets the **age** field to **31**.

9. Delete a single user document from the **users** collection:



```
db.users.deleteOne({ name: 'user2' })
```

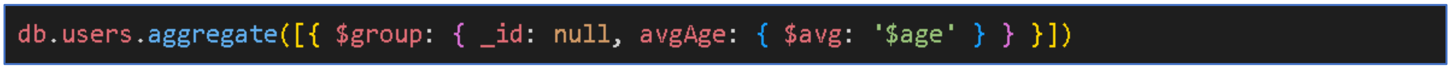
Playground output:



```
{
  "acknowledged": true,
  "deletedCount": 1
}
```

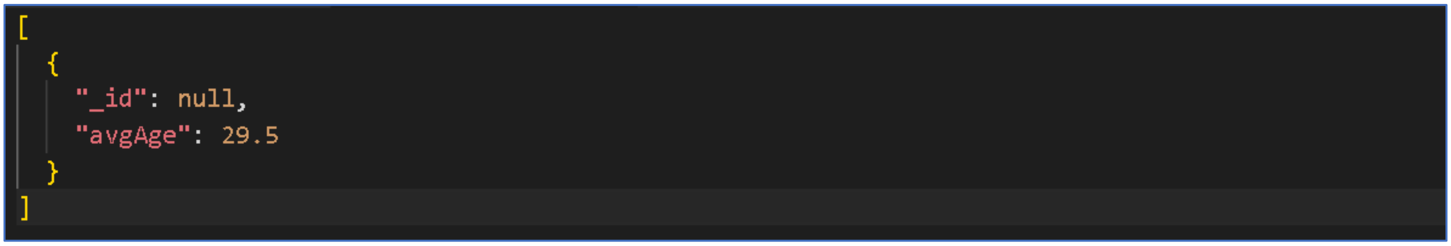
This command finds and removes the document with the name **user2** from the **users** collection.

10. Calculate the average age of the user documents in the **users** collection:



```
db.users.aggregate([{ $group: { _id: null, avgAge: { $avg: '$age' } } }])
```

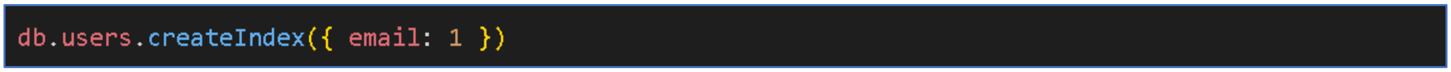
Playground output:



```
[
  {
    "_id": null,
    "avgAge": 29.5
  }
]
```

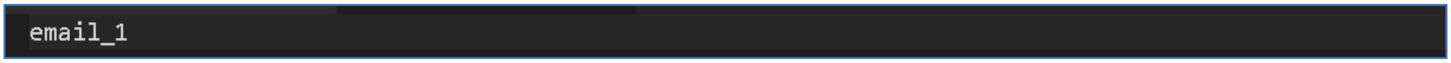
This command groups all documents due to the **`_id: null`** parameter and calculates the average age using the **`$avg`** aggregation operator on the **`age`** field.

11. Create an index for the **`email`** field in the **`users`** collection:



```
db.users.createIndex({ email: 1 })
```

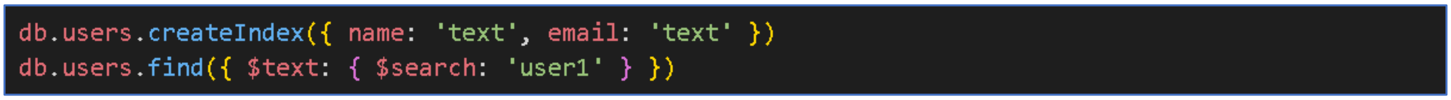
Playground output:



```
email_1
```

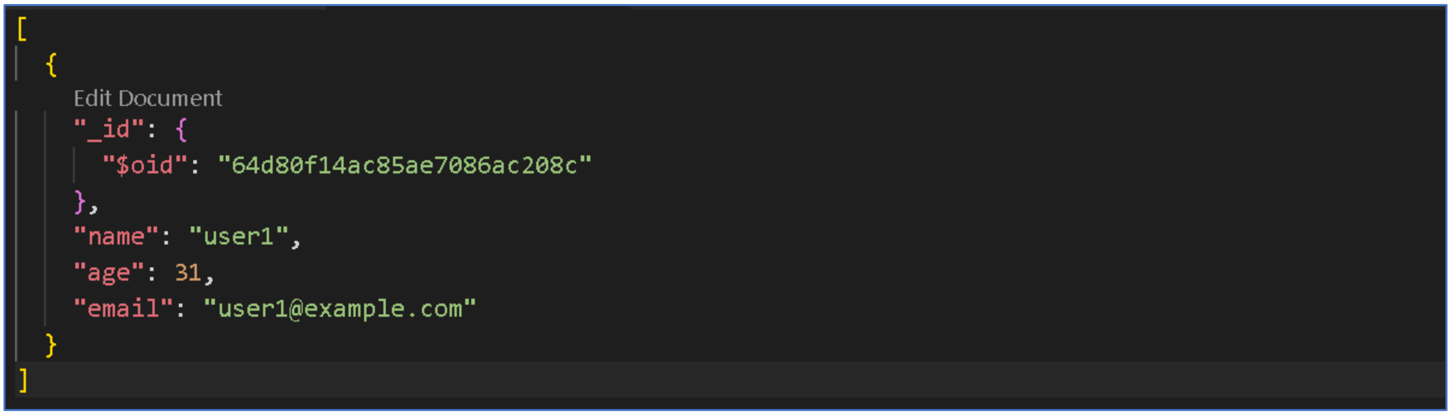
This command creates an ascending index for the **`email`** field, which enables faster searches.

12. Create a text index on the **`name`** and **`email`** fields in the **`users`** collection:



```
db.users.createIndex({ name: 'text', email: 'text' })
db.users.find({ $text: { $search: 'user1' } })
```

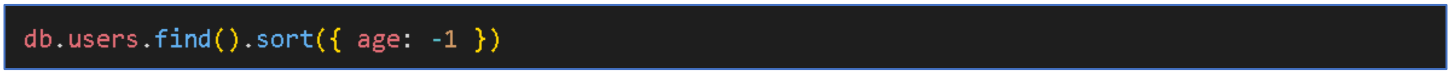
Playground output:



```
[
  {
    Edit Document
    "_id": {
      "$oid": "64d80f14ac85ae7086ac208c"
    },
    "name": "user1",
    "age": 31,
    "email": "user1@example.com"
  }
]
```

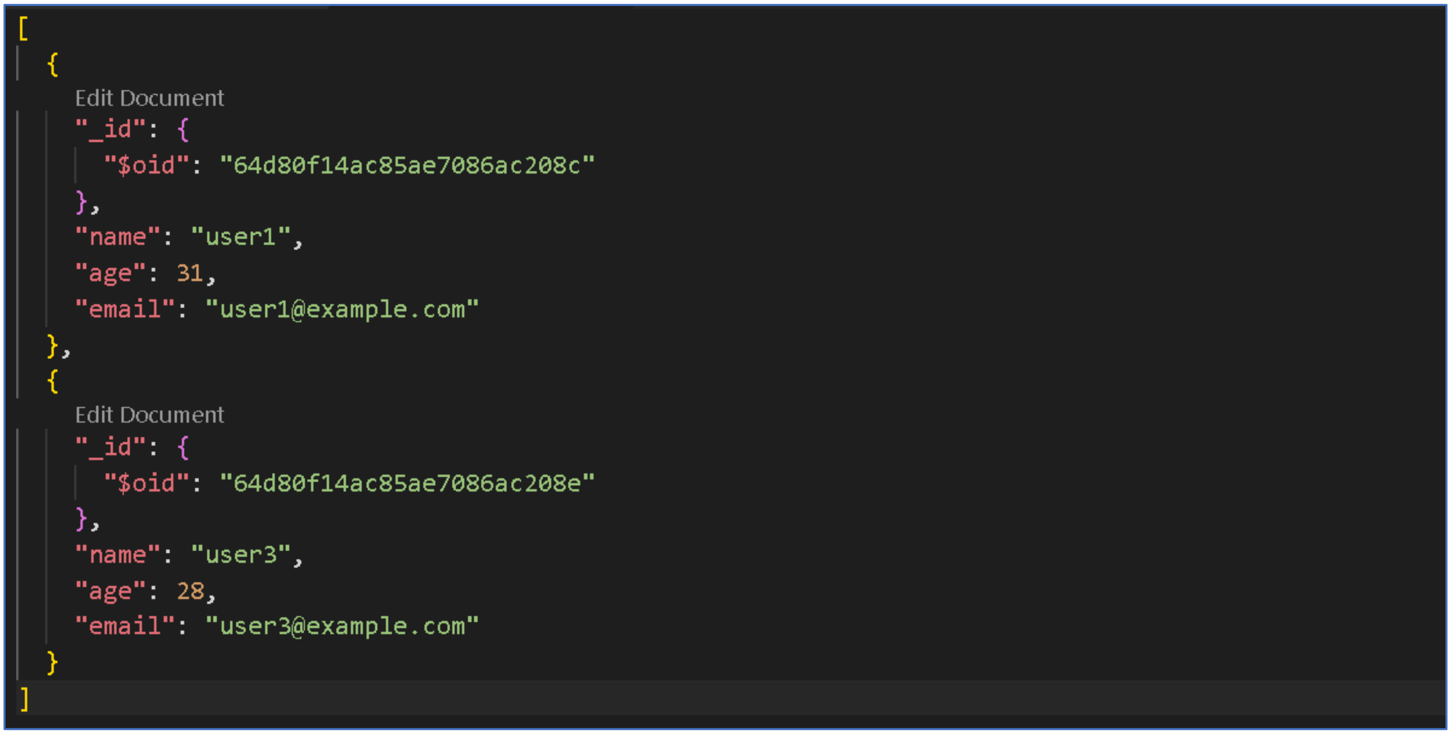
Creating a text index on the **name** and **email** fields of the **users** collection enables efficient text-based queries, such as searching for specific terms like **user1**. The command **db.users.find({ $text: { $search: 'user1' } })** is used to perform a text search on the **users** collection. When executed, it looks for documents where the text index matches the term **user1**, which could span across multiple fields in the collection.

13. Display the documents from the **users** collection in descending order by **age**:



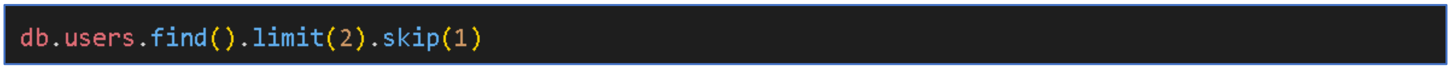
```
db.users.find().sort({ age: -1 })
```

Playground output:



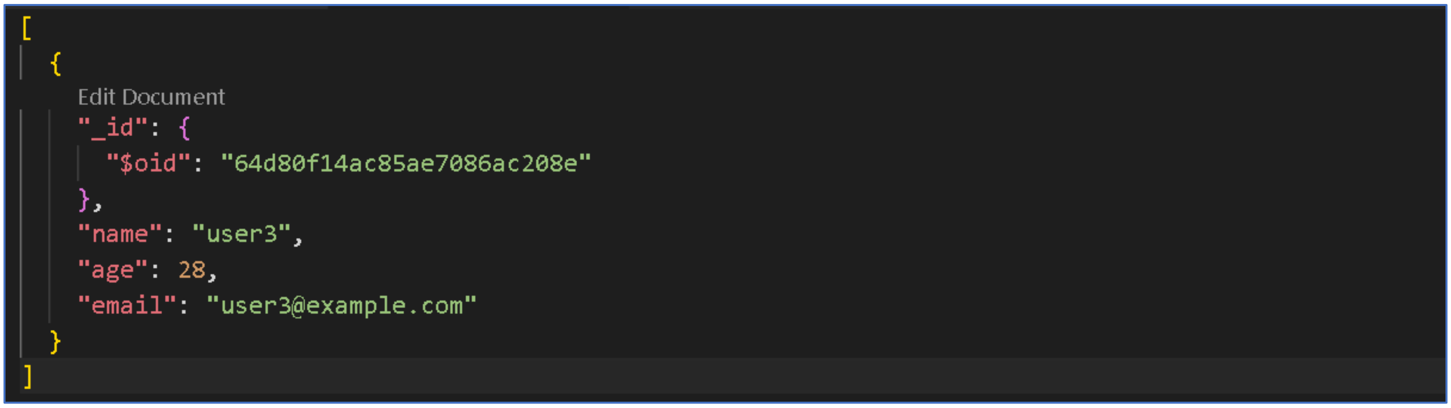
```
[
  {
    Edit Document
    "_id": {
      "$oid": "64d80f14ac85ae7086ac208c"
    },
    "name": "user1",
    "age": 31,
    "email": "user1@example.com"
  },
  {
    Edit Document
    "_id": {
      "$oid": "64d80f14ac85ae7086ac208e"
    },
    "name": "user3",
    "age": 28,
    "email": "user3@example.com"
  }
]
```

14. Display some documents from the **users** collection, skipping the first result:



```
db.users.find().limit(2).skip(1)
```

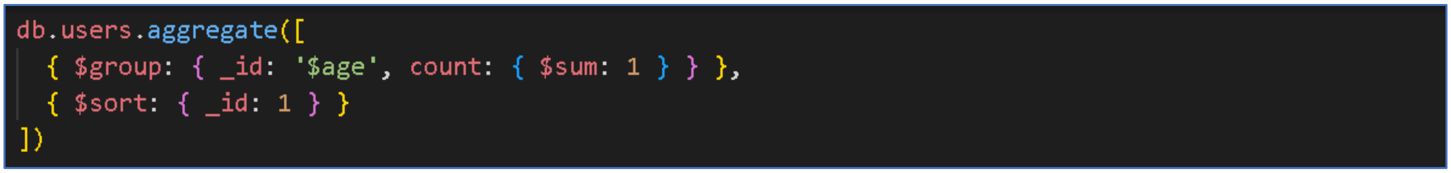
Playground output:



```
[
  {
    Edit Document
    "_id": {
      "$oid": "64d80f14ac85ae7086ac208e"
    },
    "name": "user3",
    "age": 28,
    "email": "user3@example.com"
  }
]
```

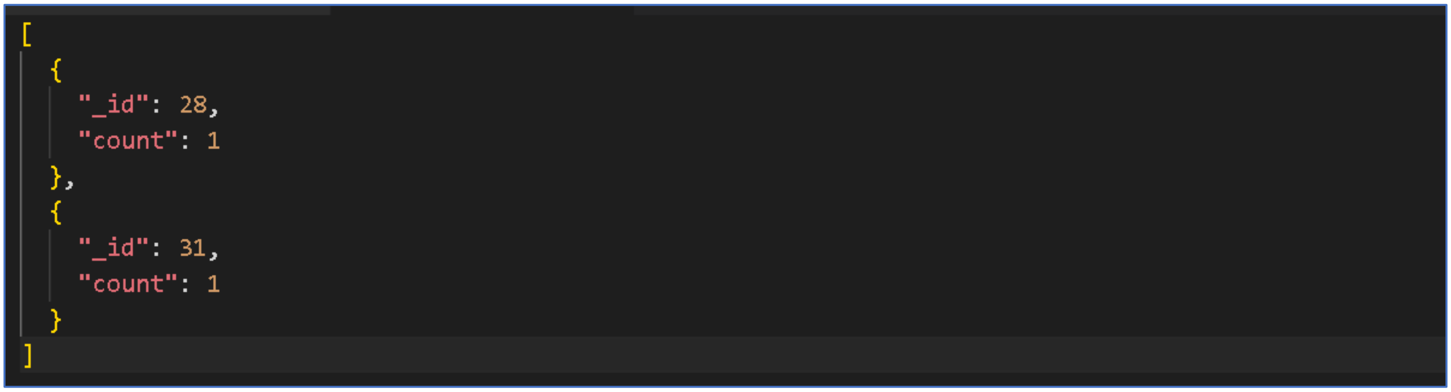
The **limit(2)** command specifies the max number of documents to return, while the **skip(1)** command tells MongoDB to skip the very first document. This approach is commonly applied when implementing pagination in database queries.

15. Group and count user documents in the **users** collection based on the **age** field:



```
db.users.aggregate([
  { $group: { _id: '$age', count: { $sum: 1 } } },
  { $sort: { _id: 1 } }
])
```

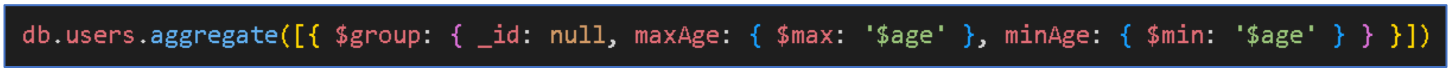
Playground output:



```
[
  {
    "_id": 28,
    "count": 1
  },
  {
    "_id": 31,
    "count": 1
  }
]
```

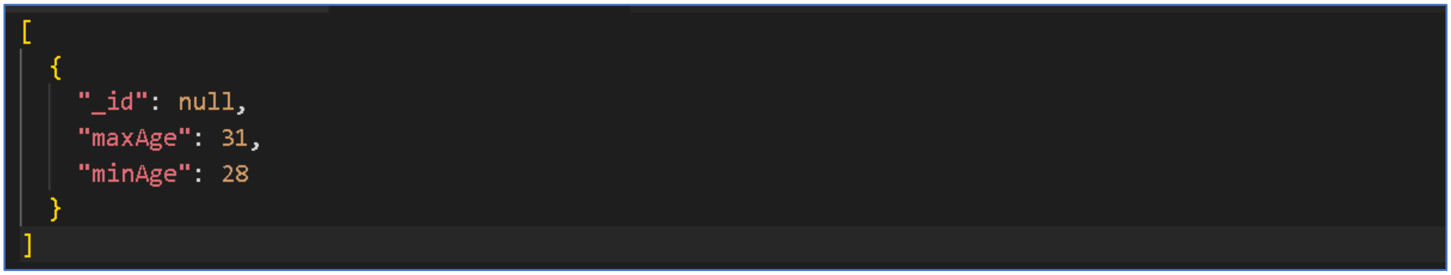
The **db.users.aggregate([...])** command, when provided with a specific aggregation pipeline, organizes user profiles from the **users** collection by their age. The **$group** stage groups the documents using the **age** field, while the **$sum** aggregation operator calculates the number of profiles for each age. Subsequently, the **$sort** stage arranges these grouped results in ascending order based on age.

16. Identify the oldest and youngest users in the **users** collection:



```
db.users.aggregate([{ $group: { _id: null, maxAge: { $max: '$age' }, minAge: { $min: '$age' } } }])
```

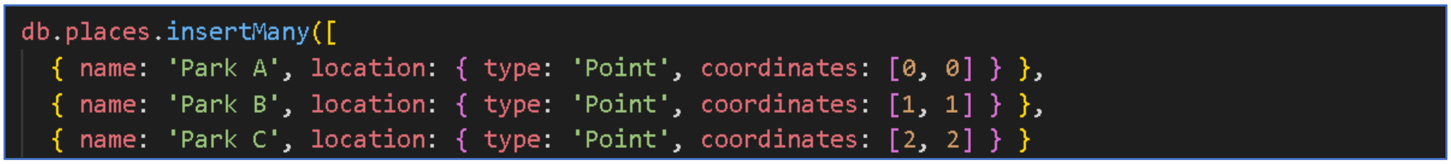
Playground output:



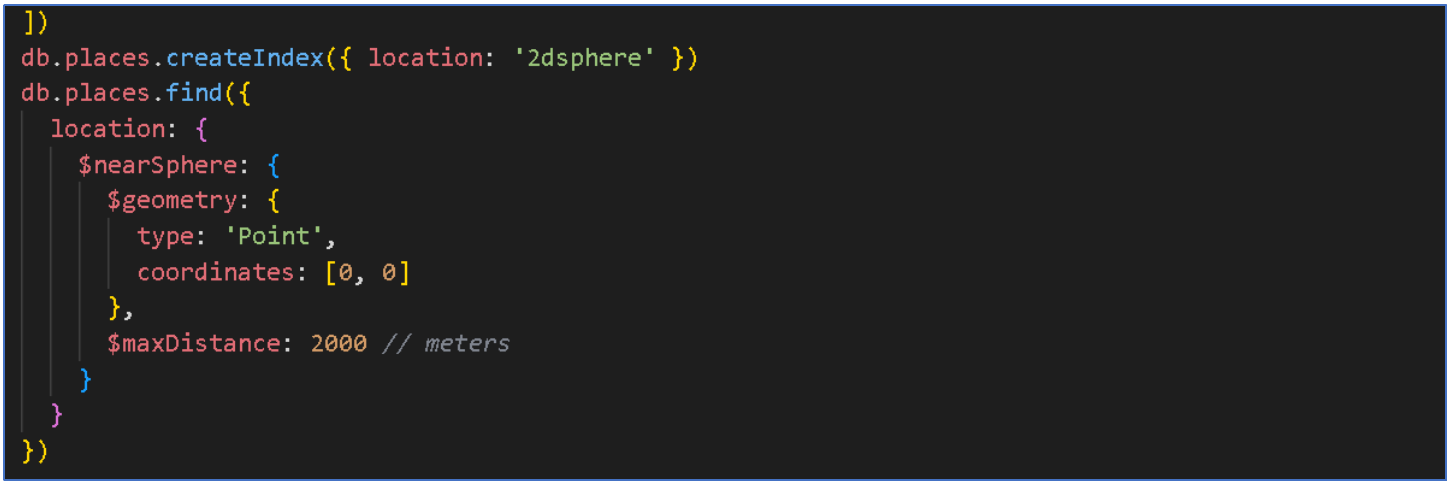
```
[
  {
    "_id": null,
    "maxAge": 31,
    "minAge": 28
  }
]
```

The provided command employs a specific aggregation pipeline to discern the ages of the oldest and youngest users from the **users** collection. Within the **$group** stage, the **$max** aggregation operator calculates the highest age, and the **$min** operator determines the lowest age. Both operations are executed without specifically grouping the data by any field. As an outcome, you receive a singular document that showcases both the calculated maximum and minimum ages.

17. Using geospatial queries, locate places within a specific radius from a given point:

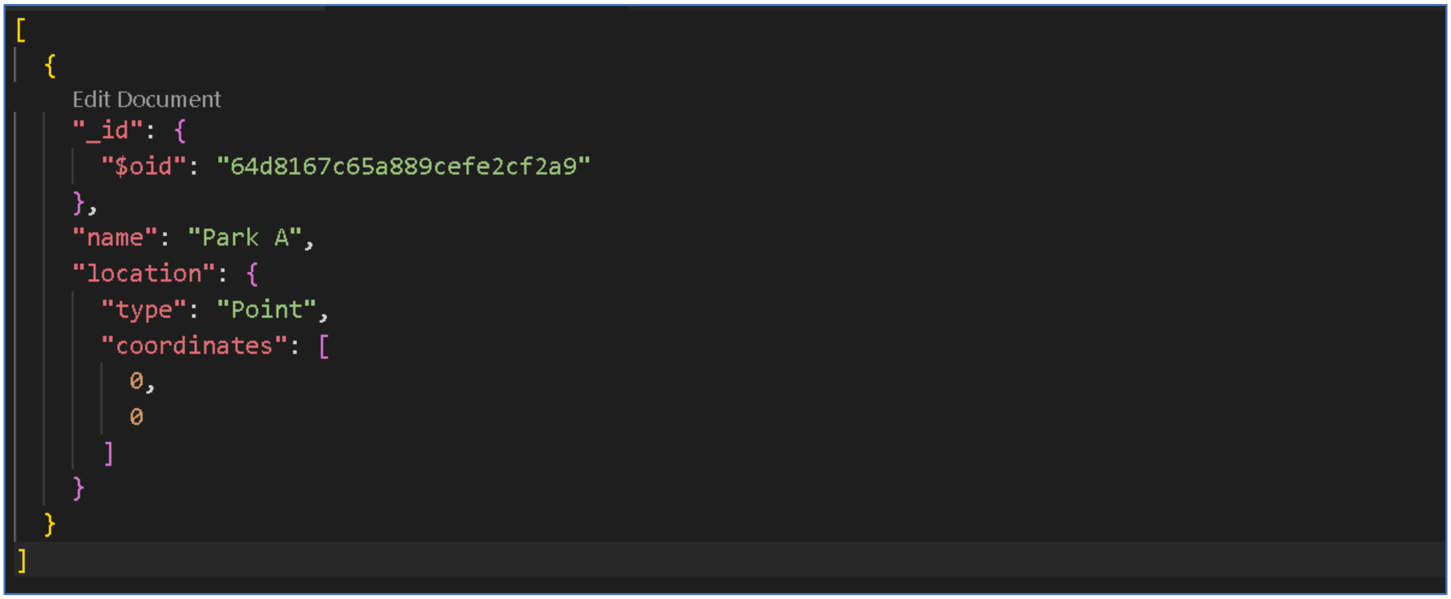


```
db.places.insertMany([
  { name: 'Park A', location: { type: 'Point', coordinates: [0, 0] } },
  { name: 'Park B', location: { type: 'Point', coordinates: [1, 1] } },
  { name: 'Park C', location: { type: 'Point', coordinates: [2, 2] } }
```



```
])
db.places.createIndex({ location: '2dsphere' })
db.places.find({
  location: {
    $nearSphere: {
      $geometry: {
        type: 'Point',
        coordinates: [0, 0]
      },
      $maxDistance: 2000 // meters
    }
  }
})
```

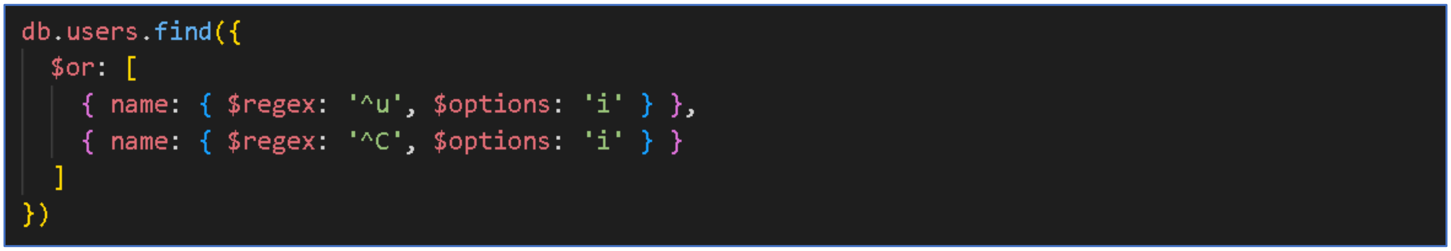
Playground output:



```
[
  {
    Edit Document
    "_id": {
      "$oid": "64d8167c65a889cefe2cf2a9"
    },
    "name": "Park A",
    "location": {
      "type": "Point",
      "coordinates": [
        0,
        0
      ]
    }
  }
]
```

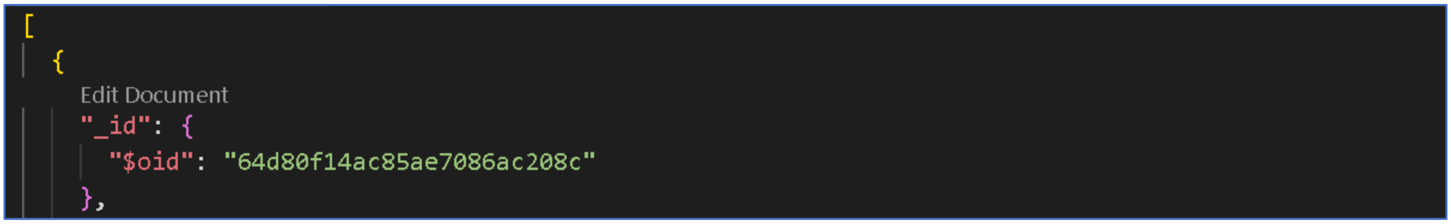
Geospatial queries can be utilized to identify places located within a certain radius from a designated point. In this context, the locations of places are represented as coordinates and are indexed using the **2dsphere** index. When the correct queries are run, they search for and retrieve locations based on the proximity criteria provided.

18. Locate users whose names start with **u** or **C**:

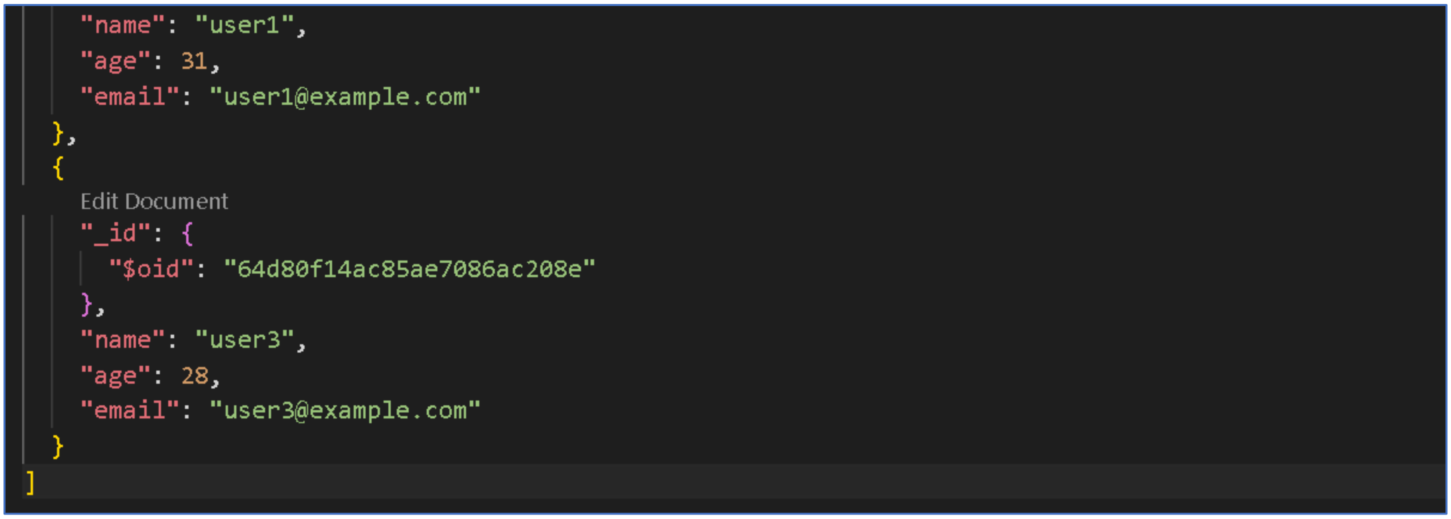


```
db.users.find({
  $or: [
    { name: { $regex: '^u', $options: 'i' } },
    { name: { $regex: '^C', $options: 'i' } }
  ]
})
```

Playground output:



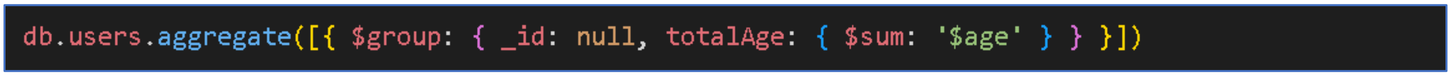
```
[
  {
    Edit Document
    "_id": {
      "$oid": "64d80f14ac85ae7086ac208c"
    },
```



```
    "name": "user1",
    "age": 31,
    "email": "user1@example.com"
  },
  {
    Edit Document
    "_id": {
      "$oid": "64d80f14ac85ae7086ac208e"
    },
    "name": "user3",
    "age": 28,
    "email": "user3@example.com"
  }
]
```

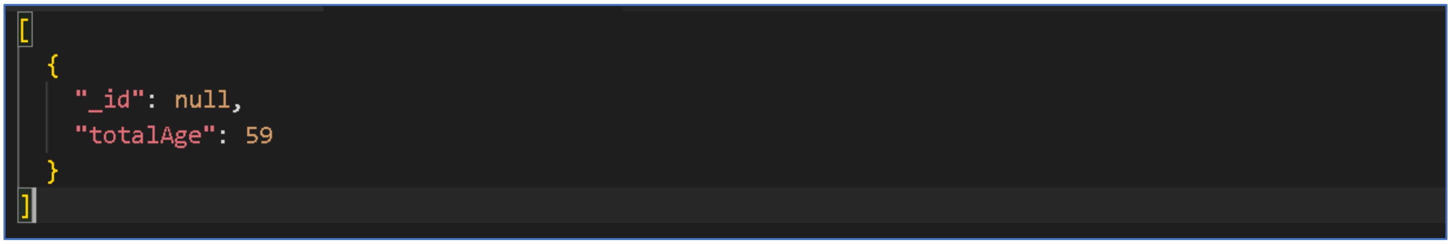
In the command above, an advanced query using the **$or** operator is combined with regular expressions. The **$options: ‘i’** line ensures case-insensitive matching, making the search process both flexible and comprehensive.

19. Calculate the total age of the users in the **users** collection:



```
db.users.aggregate([{ $group: { _id: null, totalAge: { $sum: '$age' } } }])
```

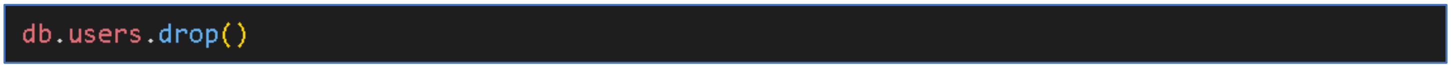
Playground output:



```
[
  {
    "_id": null,
    "totalAge": 59
  }
]
```

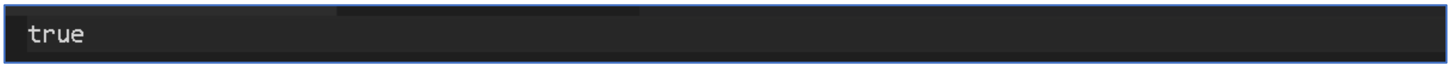
Here, all users are grouped together by passing the null identifier and their ages are summed by passing the **$sum** aggregation operator.

20. Remove the **users** collection from the database:



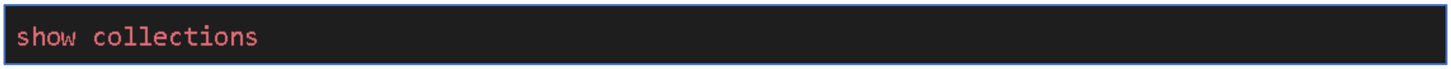
```
db.users.drop()
```

Playground output:



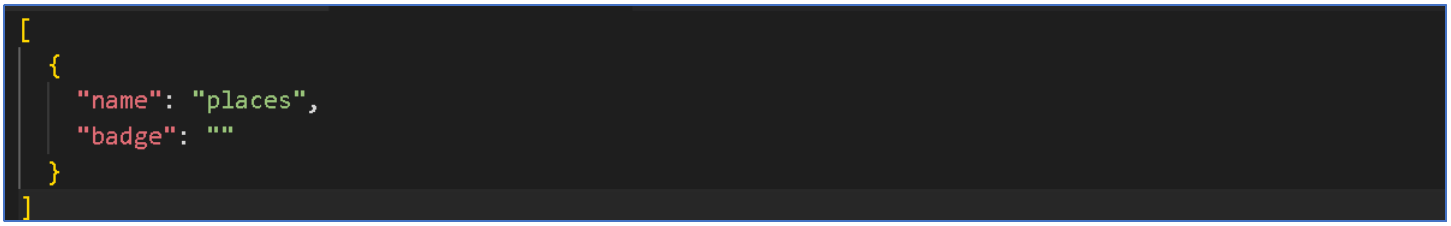
```
true
```

21. Display the list of collections available in the current database:



```
show collections
```

Playground output:



```
[
  {
    "name": "places",
    "badge": ""
  }
]
```

PUSH YOUR WORK TO GITHUB TO SUBMIT