Activity 11

Cloud Computing

Group #	9

Participating group members

Student ID	Name
6630132921	Thara Waranuset
6632045321	Chalisa Wirojratana
6632024121	Jirapat Kongkatonyoosakul

Objective

- To learn how to store and query mongodb on Cloud
- To experience how to integrate backend on EC2 to mongodb on Cloud

Submission instructions

You must submit, as a group, to MyCourseVille activity 11 assignments as followed:

- **Part 1:** follow the instructions, save screenshots, and upload worksheet
- **Part 2:** modify backend code to implement "delete" function using mongoose, ask TA to check the result
- **Part 3:** modify backend code to implement "filter-v2.0" function using mongoose, ask TA to check the result

Part 0: Preparing Tools

- Signup for mongodb atlas at https://www.mongodb.com/
- Create a cluster and configure all necessary parameters

- Install mongodb compass as a client to play with mongodb
- Deploy the provided source code of both frontend and backend servers on your EC2 instance
- More detail instructions in the activity preparation

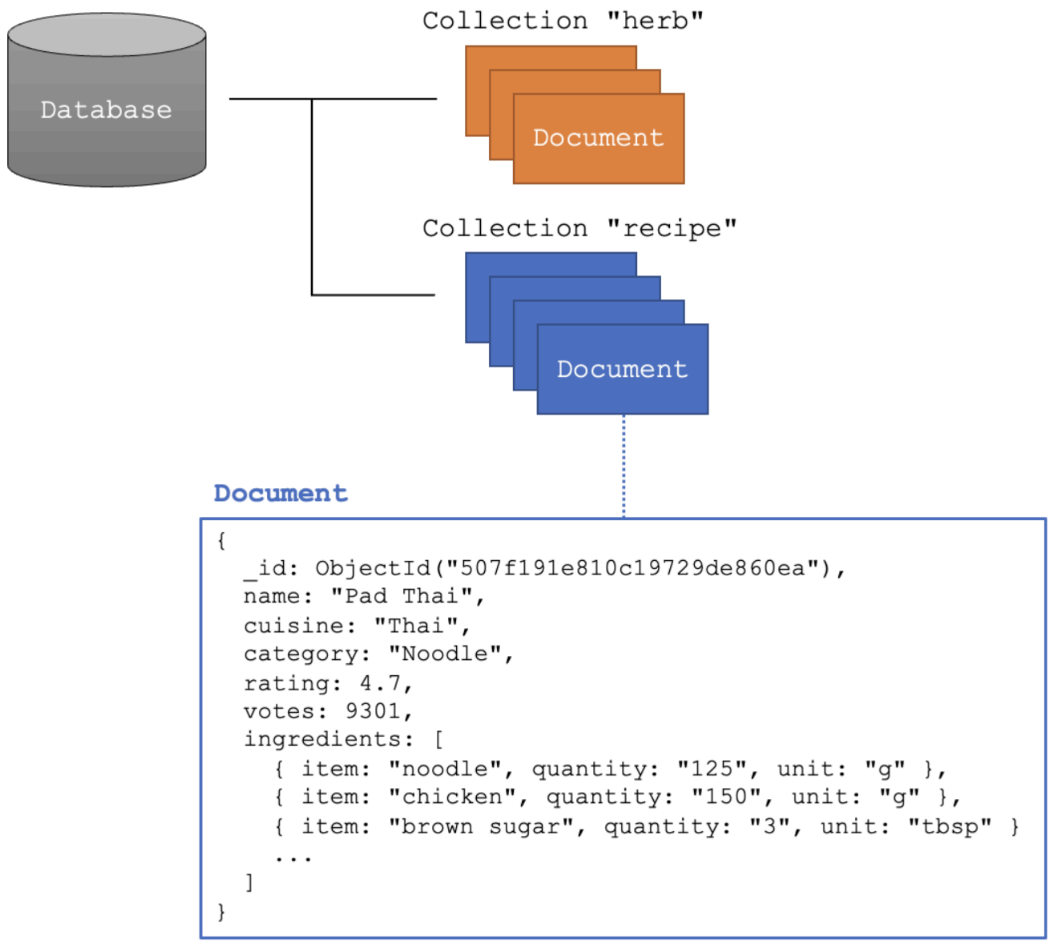
Part 1: Understand how to add and query documents

In Mongodb, each data record is called a “document”. A document contains fields with values. Each field can be a single value (e.g. string, integer), a list, or a dictionary.

```
{
  _id: ObjectId("507f191e810c19729de860ea"),
  name: "Pad Thai",
  cuisine: "Thai",
  category: "Noodle",
  rating: 4.7,
  votes: 9301,
  ingredients: [
    { item: "noodle", quantity: "125", unit: "g" },
    { item: "chicken", quantity: "150", unit: "g" },
    { item: "brown sugar", quantity: "3", unit: "tbsp" }
    ...
  ]
}
```

The above example is a document of recipes. It contains several fields including name, cuisine, category, rating, votes, and list of ingredients (item, quantity, unit). Name, cuisine, category are string; rating and votes are numbers. Ingredients is a list of items.

Mongo



```
{
  _id: ObjectId("507f191e810c19729de860ea"),
  name: "Pad Thai",
  cuisine: "Thai",
  category: "Noodle",
  rating: 4.7,
  votes: 9301,
  ingredients: [
    { item: "noodle", quantity: "125", unit: "g" },
    { item: "chicken", quantity: "150", unit: "g" },
    { item: "brown sugar", quantity: "3", unit: "tbsp" }
    ...
  ]
}
```

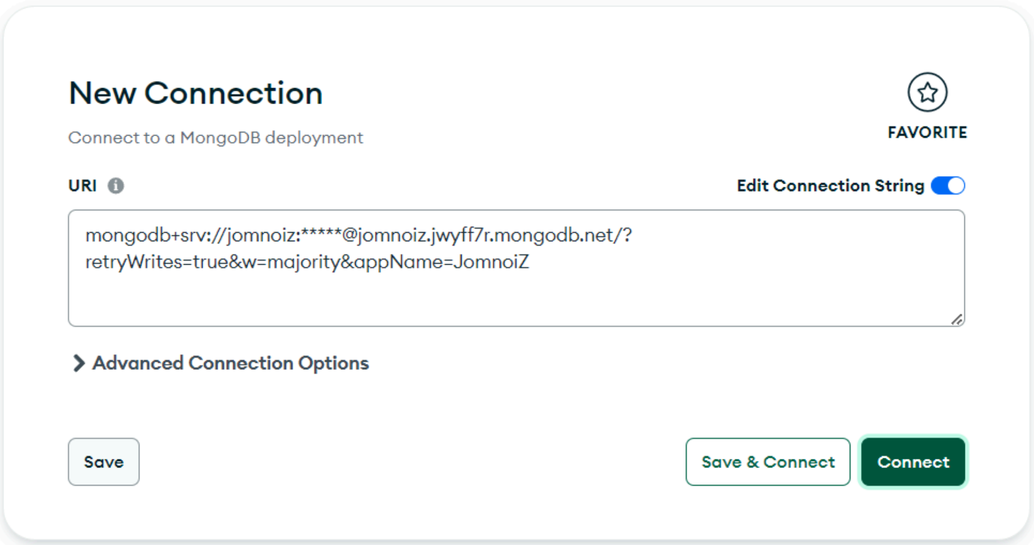
Mongodb also keeps multiple documents into “collection”. Note that each database may have multiple collections as shown in the above figure.

We can query information from mongodb using some query commands in Javascript. We will discuss about this later.

Practice with small data

Use mongodb compass and perform the following tasks:

- create new connection with your URL copied from mongodb atlas



New Connection

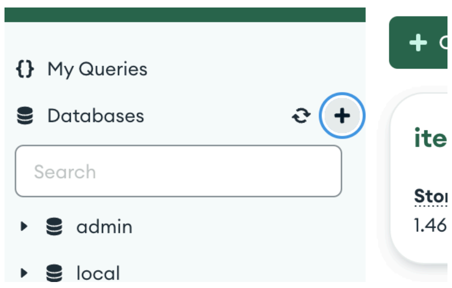
Connect to a MongoDB deployment

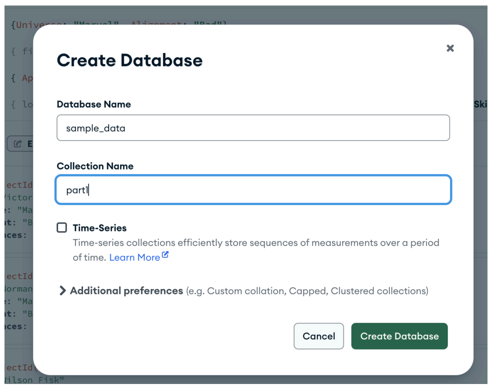
URI

Edit Connection String

```
mongodb+srv://jomnoiz:*****@jomnoiz.jwyff7r.mongodb.net/?retryWrites=true&w=majority&appName=JomnoiZ
```

- Create a new database "sample_data" and also create a collection "part1"





Create Database

Database Name: sample_data

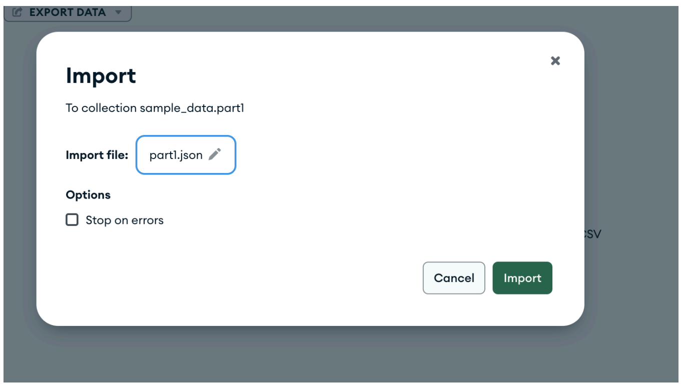
Collection Name: part1

Time-Series

Time-series collections efficiently store sequences of measurements over a period of time. Learn More

Additional preferences (e.g. Custom collation, Capped, Clustered collections)

- You must also import data from part1.json, which can be downloaded from MyCourseVille



Import

To collection sample_data.part1

Import file: part1.json

Options

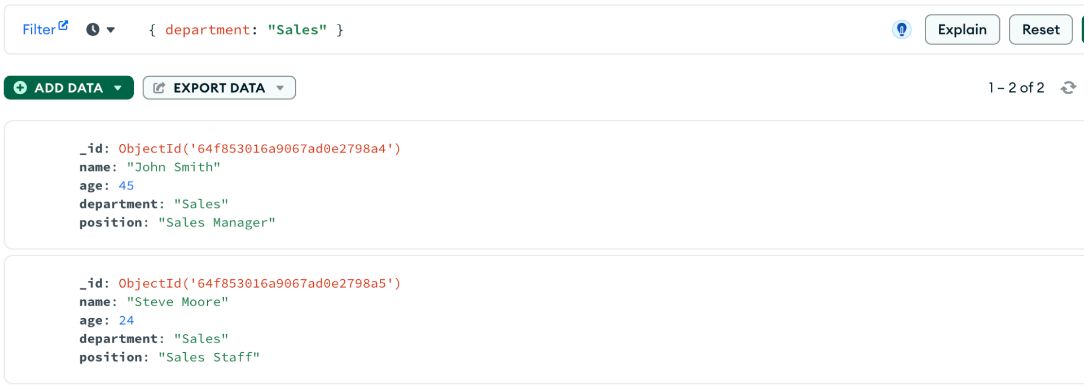
Stop on errors

Notice that the document contains name, age, department, and position.

- After importing all data, we will start with a simple query. First we will query for all staffs in Sales department by typing in the "filter" box and hit "Find":

  ```
  { department: "Sales" }
  ```

  Mongodb compass will show documents that fit the criteria (department = "Sales"). In addition, on the right hand, it shows that there are 2 documents that fit the query criteria.



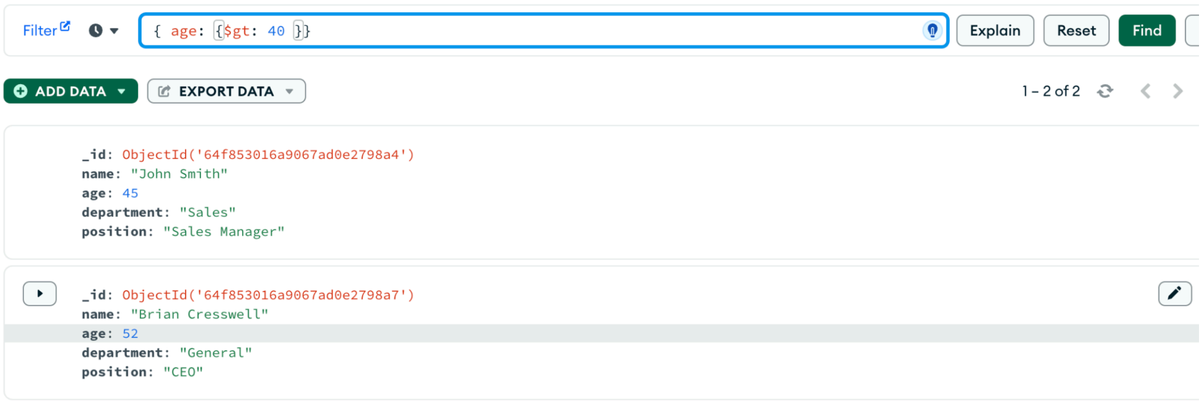
```
{ department: "Sales" }
```

```
_id: ObjectId('64f853016a9067ad0e2798a4')
name: "John Smith"
age: 45
department: "Sales"
position: "Sales Manager"
```

```
_id: ObjectId('64f853016a9067ad0e2798a5')
name: "Steve Moore"
age: 24
department: "Sales"
position: "Sales Staff"
```

- If you want to query with a numeric field (e.g. age), we can use a comparison method in the criteria. For example, to query for documents of staff's age > 40

  ```
  { age: { $gt: 40 } }
  ```



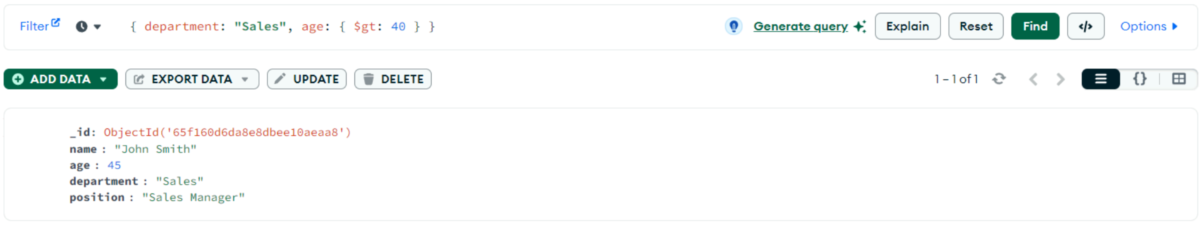
```
{ age: {$gt: 40 }}
```

```
_id: ObjectId('64f853016a9067ad0e2798a4')
name: "John Smith"
age: 45
department: "Sales"
position: "Sales Manager"
```

```
_id: ObjectId('64f853016a9067ad0e2798a7')
name: "Brian Cresswell"
age: 52
department: "General"
position: "CEO"
```

- If you want to query with and criteria e.g. staffs from department "Sales" who are more than 40 years old, you can use:

```
{ department: "Sales", age: { $gt: 40 } }
```

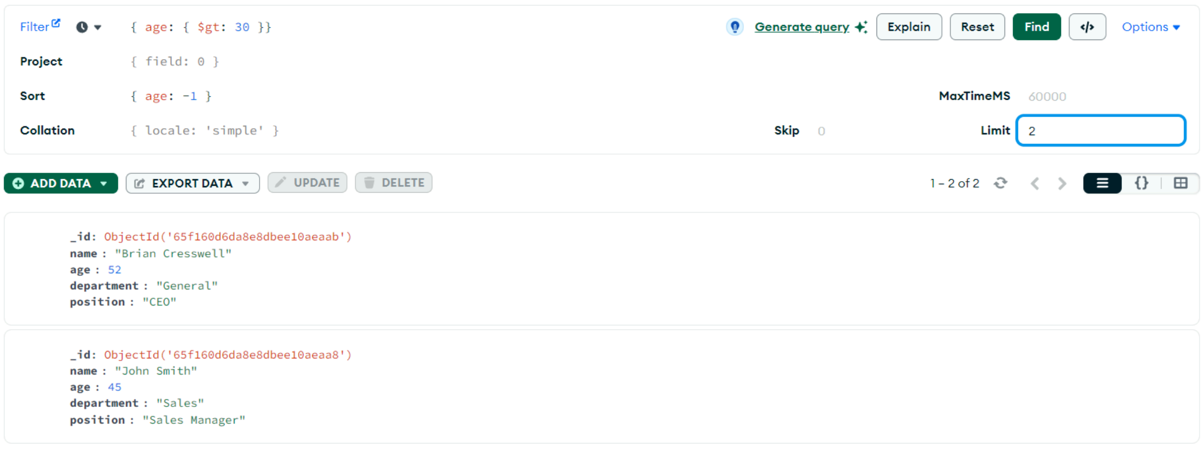


```
{ department: "Sales", age: { $gt: 40 } }
```

```
_id: ObjectId('65f160d6da8e8dbee10aeaa8')
name : "John Smith"
age : 45
department : "Sales"
position : "Sales Manager"
```

- You can query and sort in descending order (from large to small) as followed:

  Filter:
  ```
  { age: { $gt: 40 } }
  ```
  Sort:
  ```
  { age: -1 }
  ```



```
Filter     { age: { $gt: 30 }}
Project    { field: 0 }
Sort       { age: -1 }                         MaxTimeMS  60000
Collation  { locale: 'simple' }       Skip  0  Limit  2
```

```
_id: ObjectId('65f160d6da8e8dbee10aeaab')
name : "Brian Cresswell"
age : 52
department : "General"
position : "CEO"
```

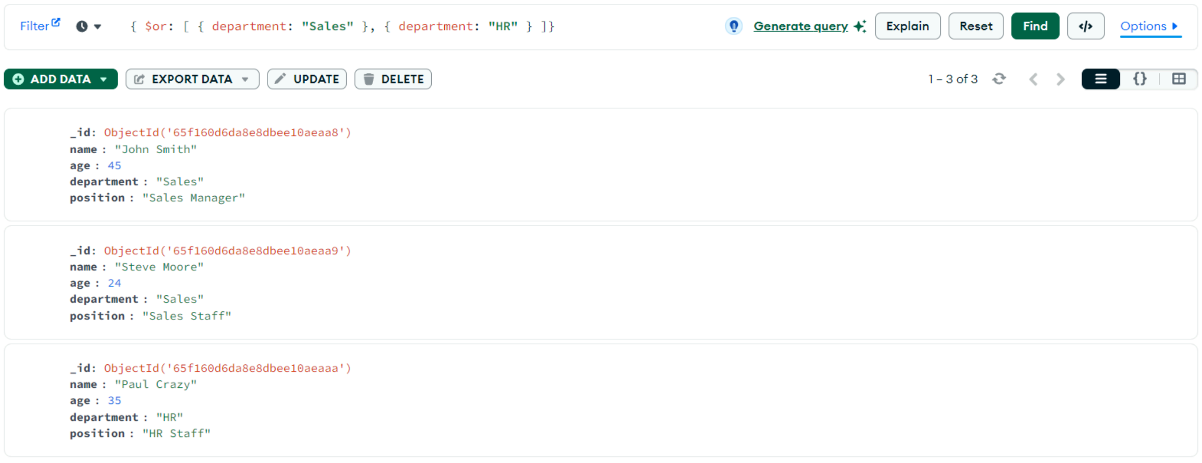
```
_id: ObjectId('65f160d6da8e8dbee10aeaa8')
name : "John Smith"
age : 45
department : "Sales"
position : "Sales Manager"
```

Note that you can enter a limit to limit the number of documents returned from the query.

- You can use "or" logic in the query. For example, if you want to query for both Sales and HR department, you can use:

  ```
  { $or: [{department: "Sales"}, {department: "HR"}] }
  ```

  Note that we use list bracket in $or operation.



```
{ $or: [ { department: "Sales" }, { department: "HR" } ]}
```

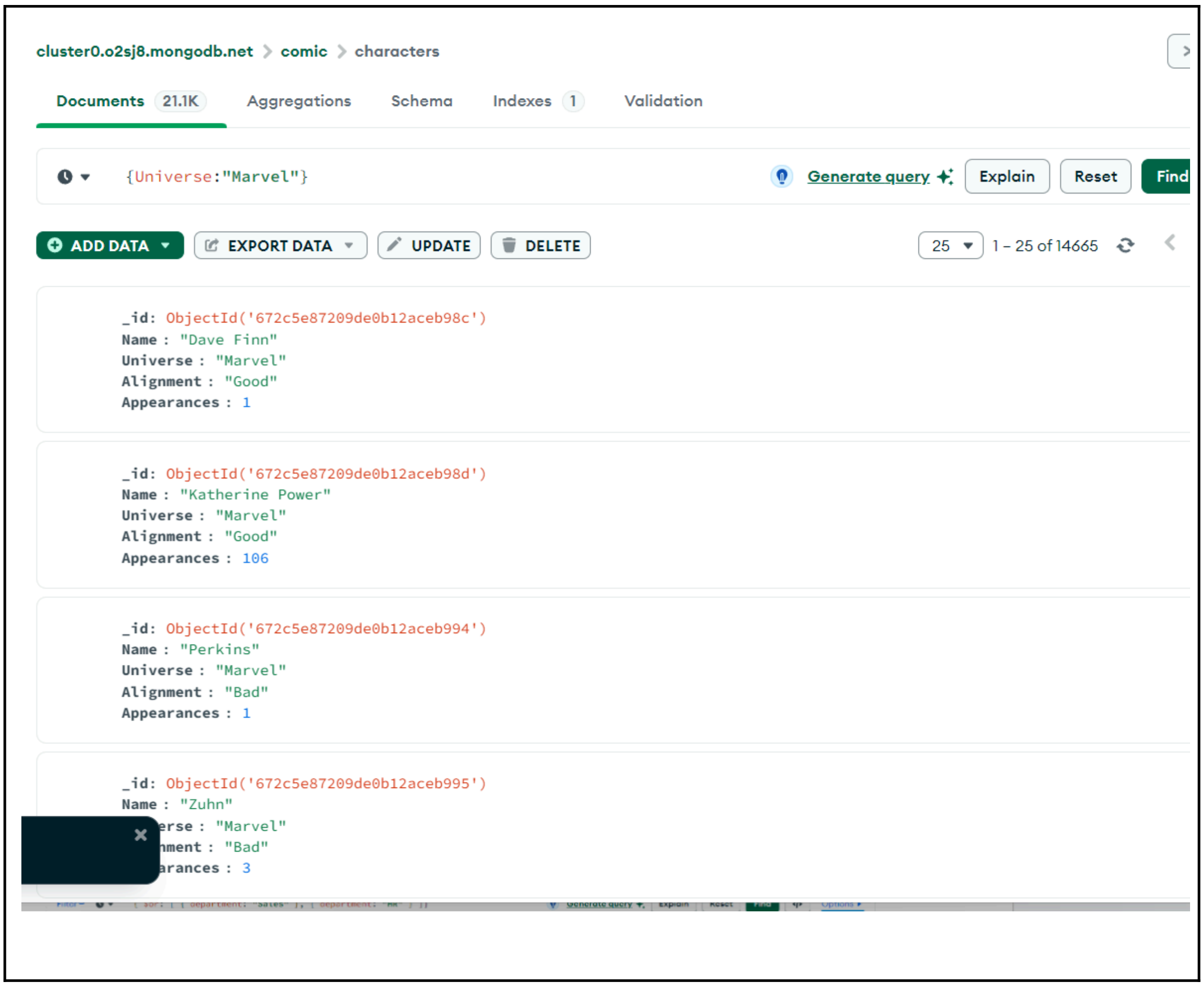
```
_id: ObjectId('65f160d6da8e8dbee10aeaa8')
name : "John Smith"
age : 45
department : "Sales"
position : "Sales Manager"

_id: ObjectId('65f160d6da8e8dbee10aeaa9')
name : "Steve Moore"
age : 24
department : "Sales"
position : "Sales Staff"

_id: ObjectId('65f160d6da8e8dbee10aeaaa')
name : "Paul Crazy"
age : 35
department : "HR"
position : "HR Staff"
```

Play with Bigger Data

- Create a new database “comic” with collection “characters”
- import characters.json into the collection
- Write the following query, capture screenshot, and save to worksheet::

All characters in Universe = “Marvel”

Capture the screenshot of your query result:

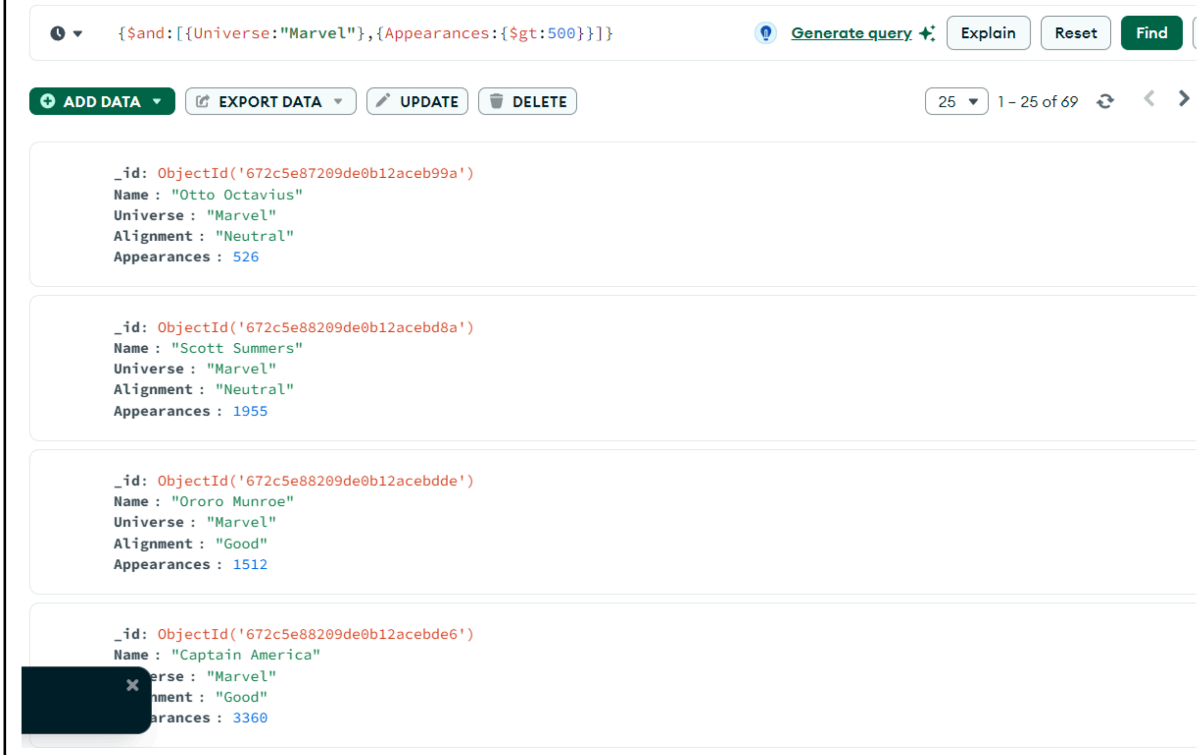


```
{Universe:"Marvel"}
```

```
_id: ObjectId('672c5e87209de0b12aceb98c')
Name : "Dave Finn"
Universe : "Marvel"
Alignment : "Good"
Appearances : 1

_id: ObjectId('672c5e87209de0b12aceb98d')
Name : "Katherine Power"
Universe : "Marvel"
Alignment : "Good"
Appearances : 106

_id: ObjectId('672c5e87209de0b12aceb994')
Name : "Perkins"
Universe : "Marvel"
Alignment : "Bad"
Appearances : 1

_id: ObjectId('672c5e87209de0b12aceb995')
Name : "Zuhn"
erse : "Marvel"
ment : "Bad"
arances : 3
```

All characters in Universe = “Marvel”, Appearances > 500

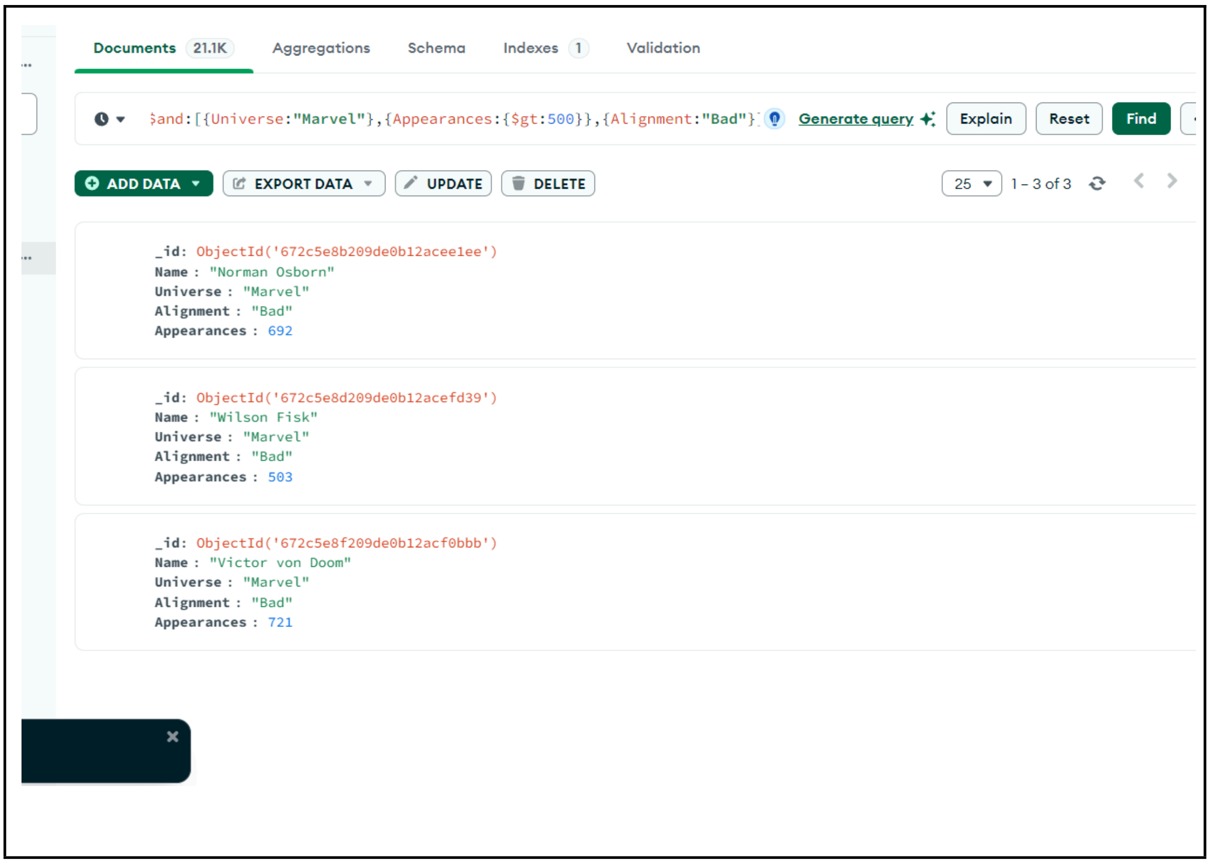
Capture the screenshot of your query result:


```
{$and:[{Universe:"Marvel"},{Appearances:{$gt:500}}]}
```

```
_id: ObjectId('672c5e87209de0b12aceb99a')
Name : "Otto Octavius"
Universe : "Marvel"
Alignment : "Neutral"
Appearances : 526

_id: ObjectId('672c5e88209de0b12acebd8a')
Name : "Scott Summers"
Universe : "Marvel"
Alignment : "Neutral"
Appearances : 1955

_id: ObjectId('672c5e88209de0b12acebdde')
Name : "Ororo Munroe"
Universe : "Marvel"
Alignment : "Good"
Appearances : 1512

_id: ObjectId('672c5e88209de0b12acebde6')
Name : "Captain America"
erse : "Marvel"
ment : "Good"
arances : 3360
```

All characters in Universe = “Marvel”, Appearances > 500, Alignment = “Bad”

Capture the screenshot of your query result:

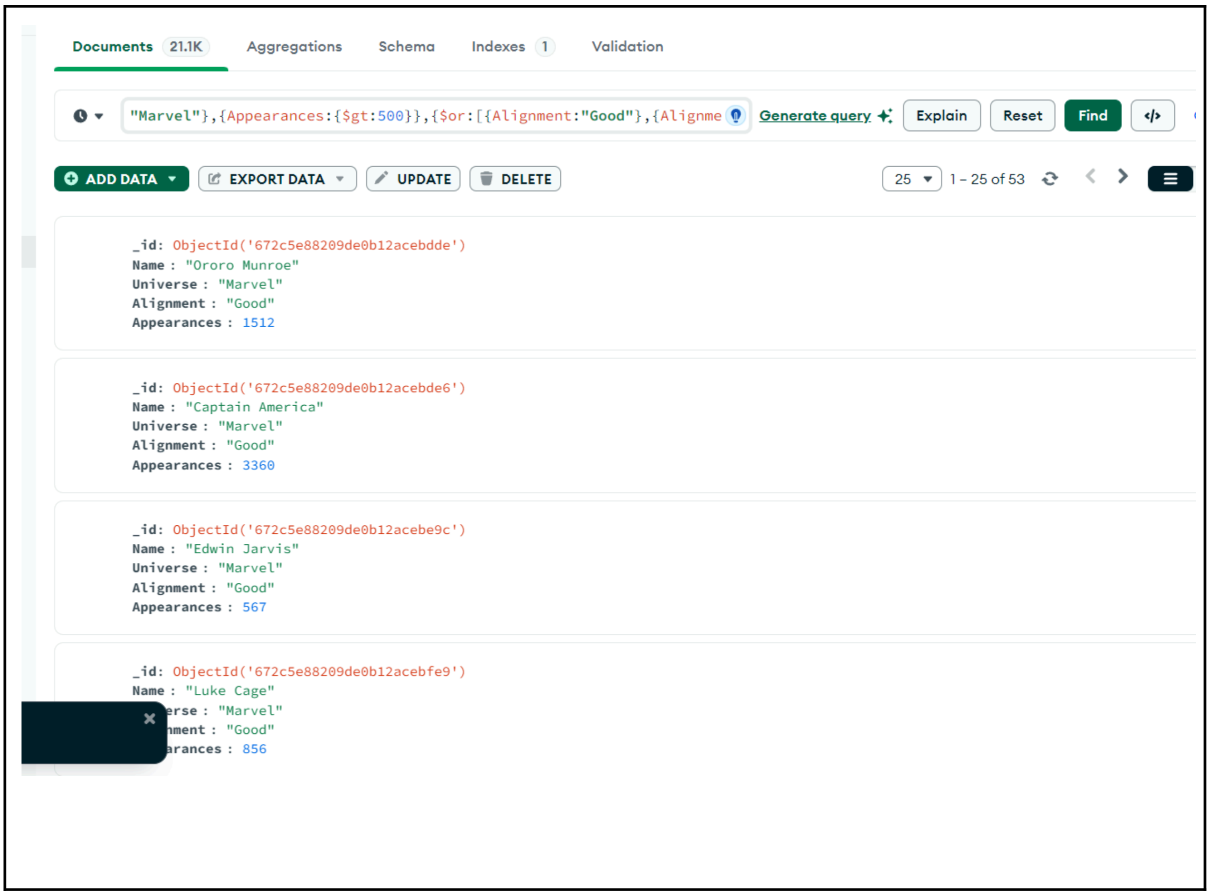


```
$and:[{Universe:"Marvel"},{Appearances:{$gt:500}},{Alignment:"Bad"}
```

```
_id: ObjectId('672c5e8b209de0b12aceelee')
Name : "Norman Osborn"
Universe : "Marvel"
Alignment : "Bad"
Appearances : 692

_id: ObjectId('672c5e8d209de0b12acefd39')
Name : "Wilson Fisk"
Universe : "Marvel"
Alignment : "Bad"
Appearances : 503

_id: ObjectId('672c5e8f209de0b12acf0bbb')
Name : "Victor von Doom"
Universe : "Marvel"
Alignment : "Bad"
Appearances : 721
```

All characters in Universe = “Marvel”, Appearances > 500, (Alignment = “Good” or Alignment = “Neutral”)

Capture the screenshot of your query result:

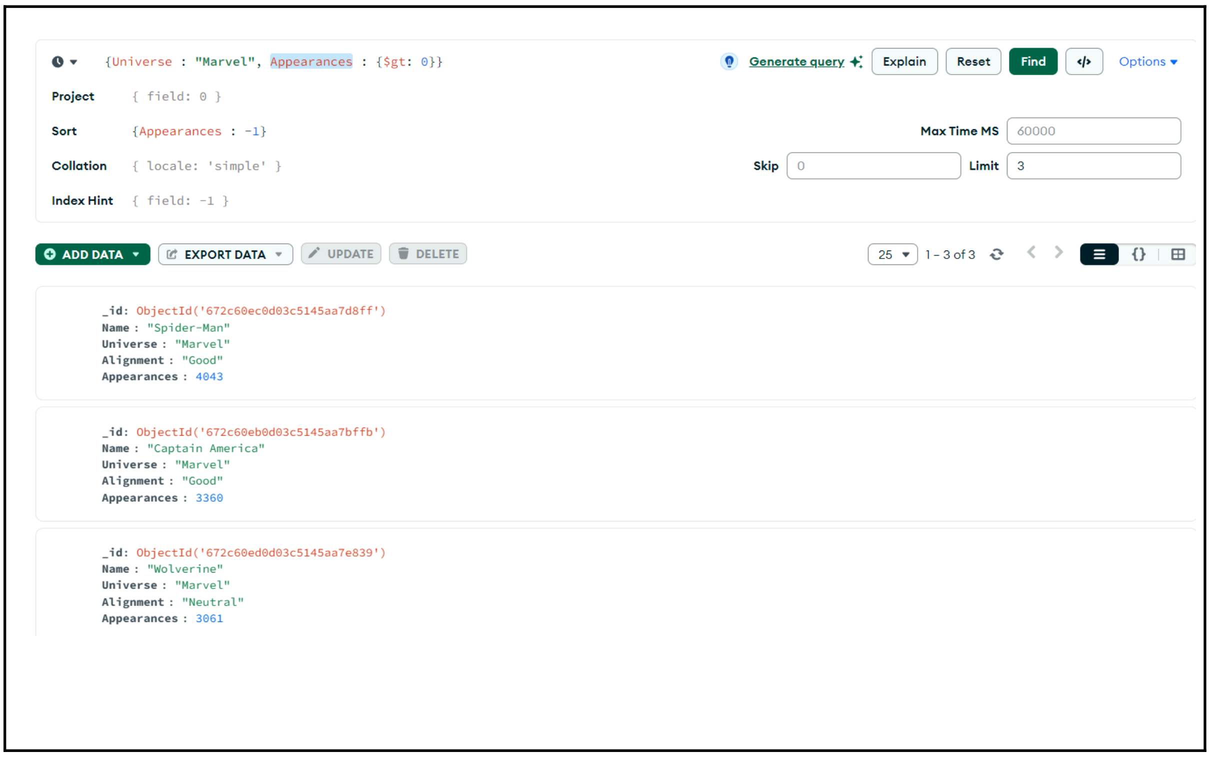


```
"Marvel"},{Appearances:{$gt:500}},{$or:[{Alignment:"Good"},{Alignme
```

```
_id: ObjectId('672c5e88209de0b12acebdde')
Name : "Ororo Munroe"
Universe : "Marvel"
Alignment : "Good"
Appearances : 1512

_id: ObjectId('672c5e88209de0b12acebde6')
Name : "Captain America"
Universe : "Marvel"
Alignment : "Good"
Appearances : 3360

_id: ObjectId('672c5e88209de0b12acebe9c')
Name : "Edwin Jarvis"
Universe : "Marvel"
Alignment : "Good"
Appearances : 567

_id: ObjectId('672c5e88209de0b12acebfe9')
Name : "Luke Cage"
   rse : "Marvel"
   ment : "Good"
   rances : 856
```

What are the top 3 characters in Universe = “Marvel” that appear the most?

Capture the screenshot of your query result:

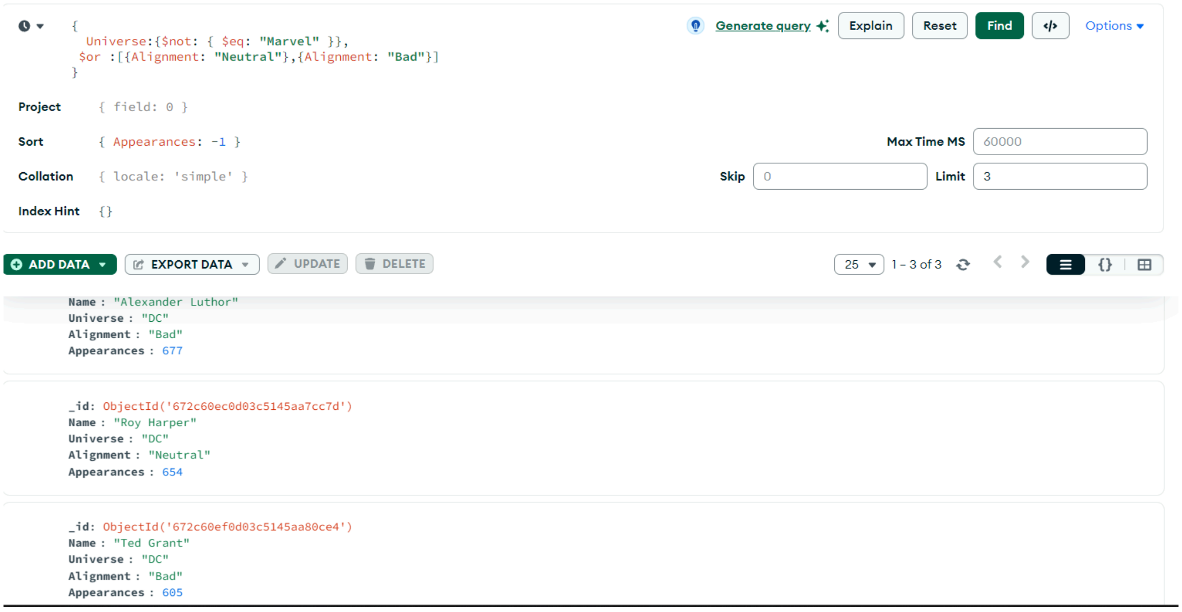


```
{Universe : "Marvel", Appearances : {$gt: 0}}
Project      { field: 0 }
Sort         {Appearances : -1}                    Max Time MS  60000
Collation    { locale: 'simple' }     Skip  0      Limit  3
Index Hint   { field: -1 }
```

```
_id: ObjectId('672c60ec0d03c5145aa7d8ff')
Name : "Spider-Man"
Universe : "Marvel"
Alignment : "Good"
Appearances : 4043

_id: ObjectId('672c60eb0d03c5145aa7bffb')
Name : "Captain America"
Universe : "Marvel"
Alignment : "Good"
Appearances : 3360

_id: ObjectId('672c60ed0d03c5145aa7e839')
Name : "Wolverine"
Universe : "Marvel"
Alignment : "Neutral"
Appearances : 3061
```

What are the top 3 characters in Universe ≠ "Marvel", (Alignment = "Neutral" or Alignment = "Bad") that appear the most?

Capture the screenshot of your query result:



```
{
  Universe:{$not: { $eq: "Marvel" }},
 $or :[{Alignment: "Neutral"},{Alignment: "Bad"}]
}
```

Project { field: 0 }

Sort { Appearances: -1 }

Max Time MS 60000

Collation { locale: 'simple' }

Skip 0

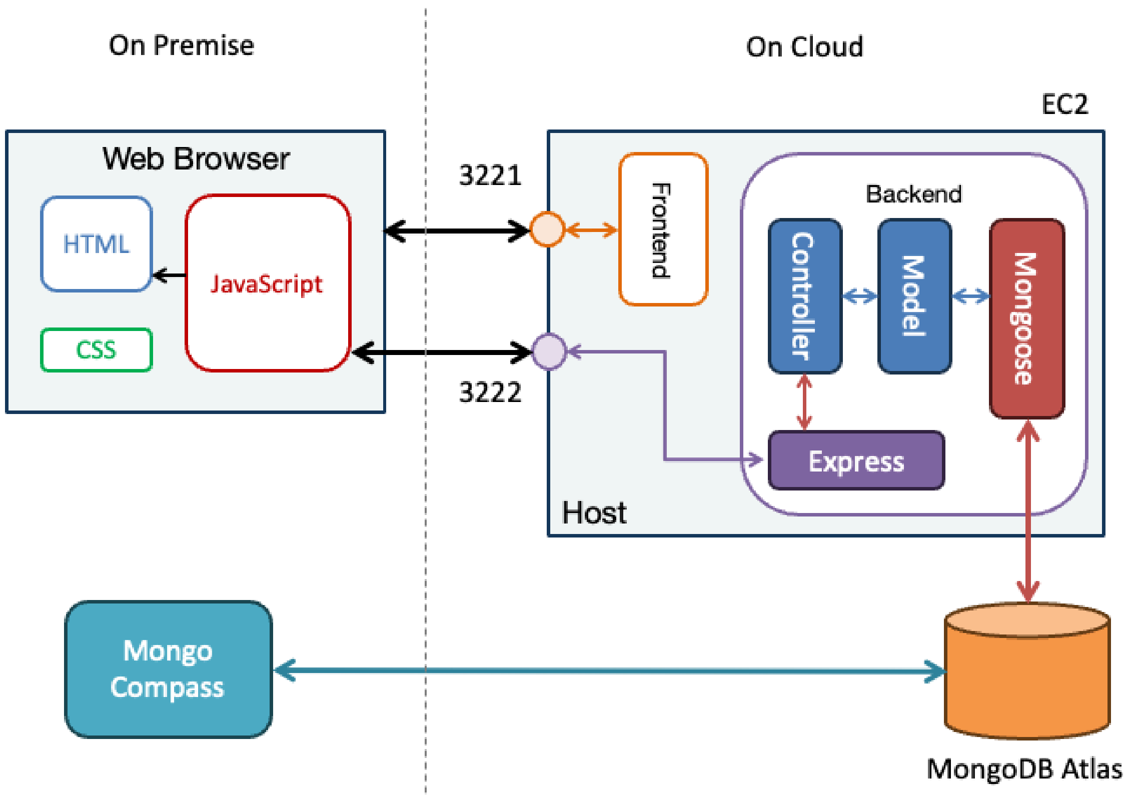
Limit 3

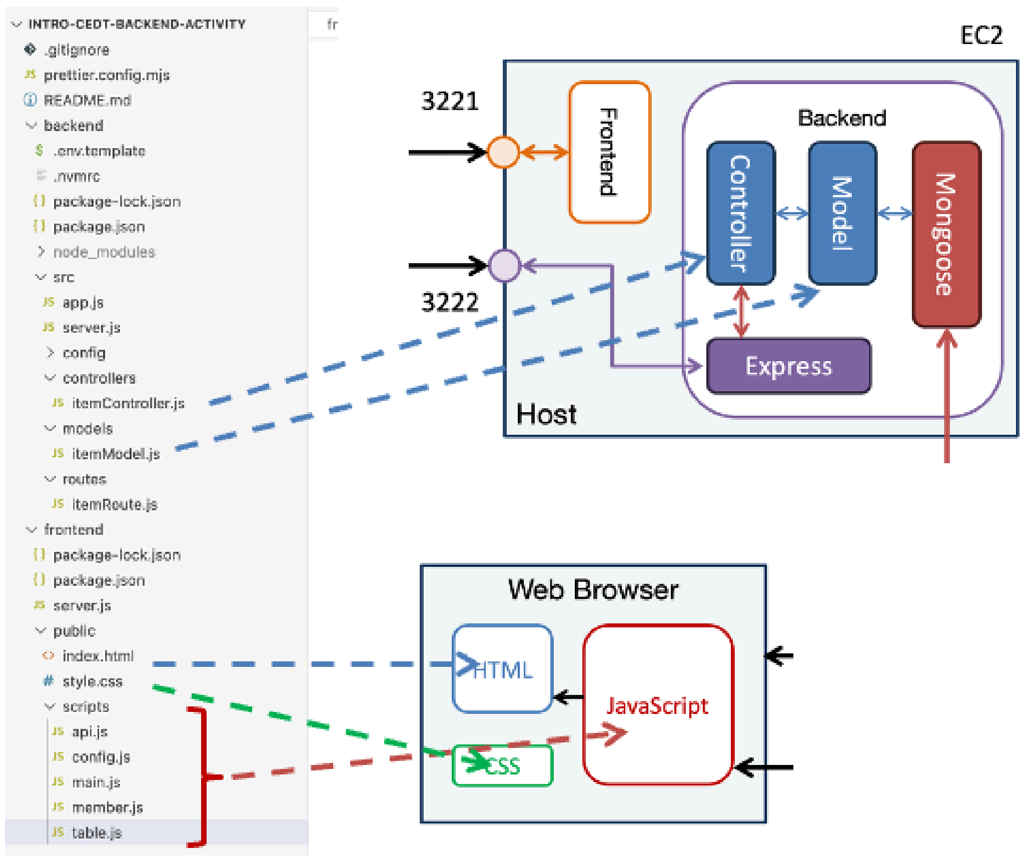
Index Hint {}

```
Name : "Alexander Luthor"
Universe : "DC"
Alignment : "Bad"
Appearances : 677

_id: ObjectId('672c60ec0d03c5145aa7cc7d')
Name : "Roy Harper"
Universe : "DC"
Alignment : "Neutral"
Appearances : 654

_id: ObjectId('672c60ef0d03c5145aa80ce4')
Name : "Ted Grant"
Universe : "DC"
Alignment : "Bad"
Appearances : 605
```

Part 2: Demonstrate the integrated application and implement a “delete” method in the endpoint

Based on the preparation, your team should be able to integrate or connect your backend to store items in the list to MongoDB atlas on Cloud. Note that the source code of the new backend that you deploy for this activity uses “Mongoose” (https://mongoosejs.com/), which is a helper library that allows JavaScript to access mongodb with ease. The architecture of the overall system is as followed:





In the starter code which can be downloaded in MyCourseVille, Mongoose is initialized in backend/src/models/itemModel.js. The backend then uses several Mongoose functions in backend./src/controllers/itemController.js (line 6, 19)..

In this part, you will have to complete a "delete" item method. Hint: you will have to modify the code in "deleteItem" function in backend/src/controllers/itemController.js

When you complete the implementation, demonstrate to the TA the following actions:

1. delete an existing item.
2. add another item.
3. delete the added item.
4. while performing each step, show the documents in the collection using mongodb compass to the TA.

If everything works, congratulations! You have successfully implemented the delete method!!

Part 3: Implement a “filter” method in the endpoint

This part is quite similar to the recent activity in the outstanding part. However, the new filter function in this activity has a “price range” which can make you also filter items using the price conditions.
Note that after you have filtered items, if you refresh the website, add new items, or delete an existing item, the filter condition will automatically change back to the original condition (i.e., name = “-- ทั้งหมด --” and no price range). Therefore, you don’t need to care about holding the filter condition after doing any actions.

In this part, you will have to complete the following steps:

1. select which http method you want to use.
2. create a new route using that http method for filter function (Implement in itemRoute.js).
3. implement the filter function in itemController.js at line 30 (backend).
4. implement filterItems function in api.js at line 25 (backend)

HINT: You can search for “TODO3” (use Ctrl+Shift+F in VSCode and type “TODO3”) in vscode to see where you have to modify to complete the filter function.

*Warning! You are allowed to query items by using mongoose only (i.e., querying all items and filtering them afterward are not allowed in this activity). Otherwise, **your score in this part will be punished by -0.5 scores**.

When you complete the implementation, demonstrate to the TA the following actions:

1. filter to “-- ทั้งหมด --" and don’t fill the price range.
2. filter to each person in the group.
3. fill some price range and filter to each person in the group again.
4. add some items and filter to each person in the group again.
5. delete some items and filter to each person in the group again.

If everything works, congratulations! You have successfully implemented the delete method!!

Outstanding: Implement a member management system.

Now, you have to create a “Member Management System” that you will be able to “add” and “delete” members and the items that are occupied by the deleted member also should be deleted.
Note that you have to store the members data in mongodb like items.

Your task is to do the following steps:

1. create the member model in memberModel.js (backend)
2. implement the create, get, and delete function in memberController.js (backend)
3. implement the create, get, and delete function in api.js (frontend)

HINT: You can search for “TODO4” in vscode to see where you have to modify to complete the member management system.

When you complete the implementation, demonstrate to the TA the following actions:

1. add new members.
2. add new items with the added members.
3. delete the added member and the added items should be removed too.
4. filter the added members in both 2. and 3.
5. while performing each step, show the documents in the collection using mongodb compass to the TA.

If everything works, congratulations! You have earned yourself the Outstanding factor!!!