3DM – MongoDB Exercise 3

Finding beautiful hikes is a challenge. While there are countless guidebooks and websites, there isn't a truly comprehensive collection. That's why you want to develop an app that solves this problem. During your research, you came across the "Innerschweizer Wanderfreunde". This association records all hikes of its members since 1990. For each hike, the starting point, destination, distance, altitude difference, month and year are recorded (see example below). The association is enthusiastic about your idea and provides you with the data set free of charge.

```
{
  "from_name": "Vellerat",
  "to_name": "Gempen",
  "year": 2016,
  "month": 11,
  "distance": 11,
  "altitude_diff": 118
}
```

The localities indicated in the data set are often small and little known. You therefore want to use the canton of the starting point as additional parameter to simplify the selection of a hike. Fortunately, you found a dataset that includes the municipality and canton for all Swiss localities. It has the following structure:

```
{
  "sprache": "fr",
  "ortschaft": "Vellerat",
  "gemeinde": "Courrendlin",
  "kanton": "JU"
}
```

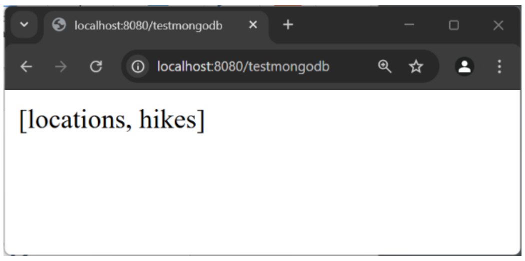
The entry `ortschaft` corresponds with the field `from_name` in the first data set.

You decide to start your project by implementing the backend. It should allow to filter hikes by the approximate distance and by the canton where the hike starts. Examples:

http://localhost:8080/api/hikes	Returns 3 random hikes
http://localhost:8080/api/hikes?distance=12	Returns 3 random hikes with a length of 12+/-2 km
http://localhost:8080/api/hikes?distance=12&canton=BE	Returns 3 random hikes with a length of 12+/-2 km starting in the canton of Berne (BE)

a) Open Compass and create a new database named «hikefinder». Add two collections named «hikes» and «locations». Import the dataset from Moodle.

b) Follow the recipe in «Spring Boot Project Setup» to create a new spring boot project. Use **hikefinder** as artifact id. Verify that get the right collection http://localhost:8080/testmongodb



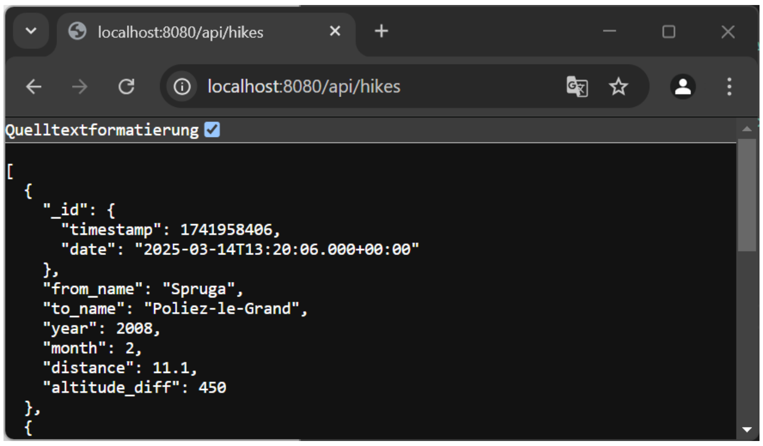
```
[locations, hikes]
```

c) In compass, design an aggregation pipeline that returns 3 random hikes.

d) Add the following endpoint to MongoDB.java and insert the pipeline from compass.

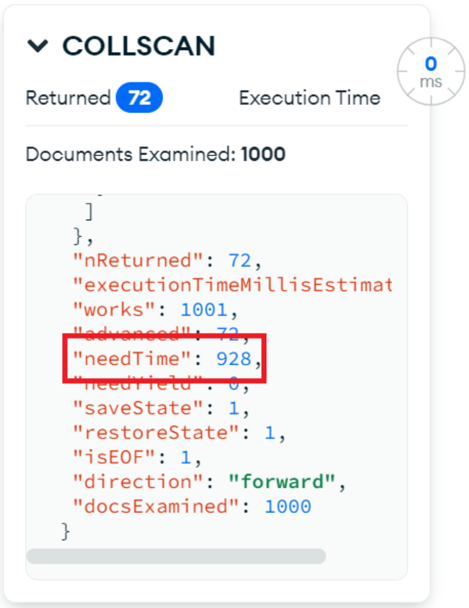
```java
@GetMapping("/api/hikes")
public ResponseEntity<Object> getHikes() {
    try {
        MongoCollection<Document> hikeCol = myDB.getCollection("hikes");
        AggregateIterable<Document> result = null;

        // aggregation pipeline
        result = hikeCol.aggregate(...);

        ArrayList<Document> hikes = result.into(new ArrayList<Document>());
        return ResponseEntity.ok(hikes);
    } catch (Exception e) {
        return ResponseEntity.status(500).build();
    }
}
```

From http://localhost:8080/api/hikes you should now get 3 random hikes. Note: How these hikes are displayed, depends on the browser.



```
[
  {
    "_id": {
      "timestamp": 1741958406,
      "date": "2025-03-14T13:20:06.000+00:00"
    },
    "from_name": "Spruga",
    "to_name": "Poliez-le-Grand",
    "year": 2008,
    "month": 2,
    "distance": 11.1,
    "altitude_diff": 450
  },
  {
```

e) Design a new query in compass that filters for hikes with a distance from 14 to 18 km. Measure the execution time of this query (with Explain). Check how many documents were examined and how many indexes have been used.

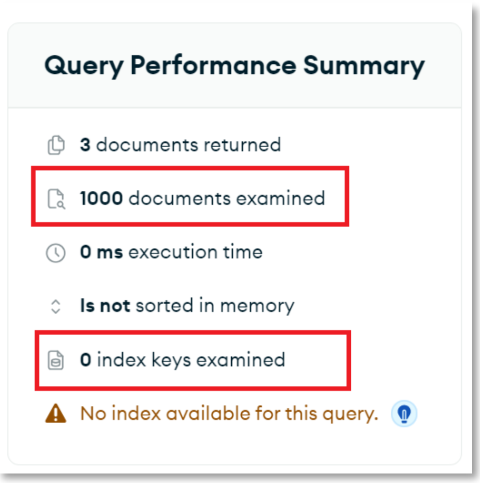


COLLSCAN

Returned 72 Execution Time 0 ms

Documents Examined: 1000

```
    ]
  },
  "nReturned": 72,
  "executionTimeMillisEstimat
  "works": 1001,
  "advanced": 72,
  "needTime": 928,
  "needYield": 0,
  "saveState": 1,
  "restoreState": 1,
  "isEOF": 1,
  "direction": "forward",
  "docsExamined": 1000
}
```



Query Performance Summary

3 documents returned

1000 documents examined

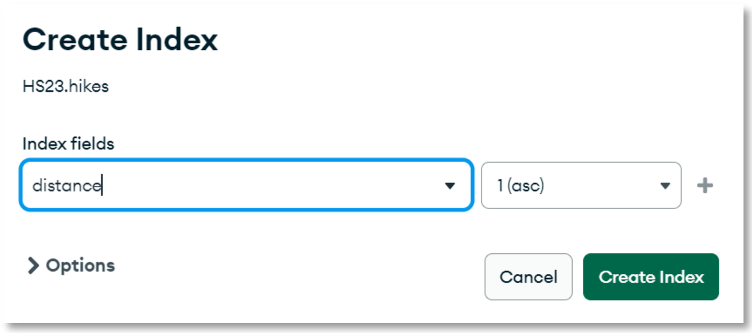
0 ms execution time

Is not sorted in memory

0 index keys examined

No index available for this query.

f) To speed up this query, add an index for the distance. Repeat the measurement and compare it with the previous result.



Create Index

HS23.hikes

Index fields

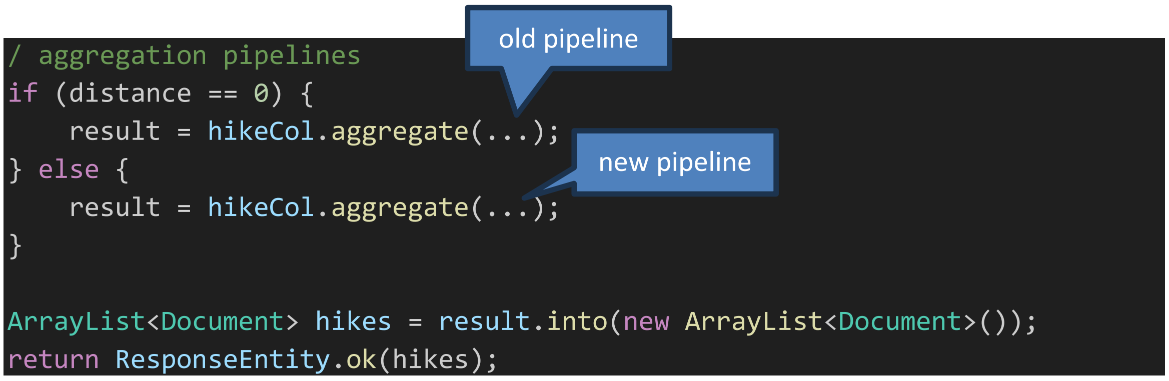
distance | 1 (asc)

g) Add an additional stage to the pipeline that randomly selects 3 hikes from the filtered result.

h) Extend the existing endpoint with an optional request parameter for the distance.

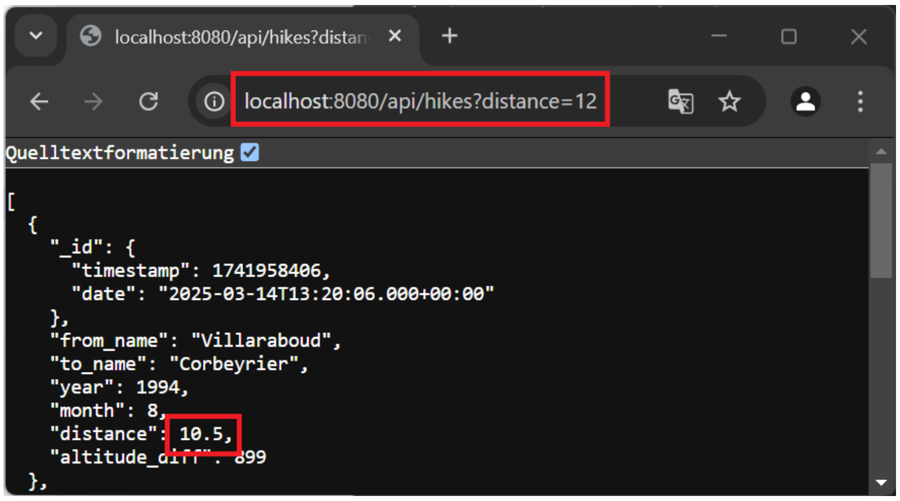
```
@GetMapping("/api/hikes")
public ResponseEntity<Object> getHikes(@RequestParam(defaultValue = "0") int distance) {
    try {
        MongoCollection<Document> hikeCol = myDB.getCollection("hikes");
        AggregateIterable<Document> result = null;
```

i) Depending on the existence of the parameter «distance», the endpoint must select the right pipeline. The code below shows parts of a possible solution. Make use of the pipeline designed in step g). Replace the numbers 14 and 18 with a distance computed from the request parameter by adding (+/- 2).



```
/ aggregation pipelines
if (distance == 0) {
    result = hikeCol.aggregate(...);
} else {
    result = hikeCol.aggregate(...);
}

ArrayList<Document> hikes = result.into(new ArrayList<Document>());
return ResponseEntity.ok(hikes);
```

It should still be possible to read 3 random hikes from http://localhost:8080/api/hikes. When adding a request parameter for the distance, the response should only contain hikes with the desired length. Examples: http://localhost:8080/api/hikes?distance=12

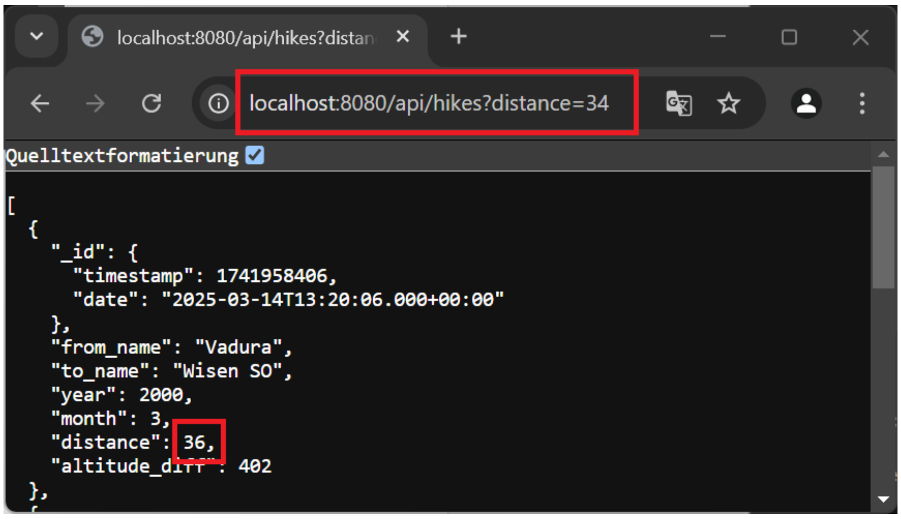


localhost:8080/api/hikes?distance=12

Quelltextformatierung ✓

```
[
  {
    "_id": {
      "timestamp": 1741958406,
      "date": "2025-03-14T13:20:06.000+00:00"
    },
    "from_name": "Villaraboud",
    "to_name": "Corbeyrier",
    "year": 1994,
    "month": 8,
    "distance": 10.5,
    "altitude_diff": 899
  },
```

http://localhost:8080/api/hikes?distance=34



localhost:8080/api/hikes?distance=34

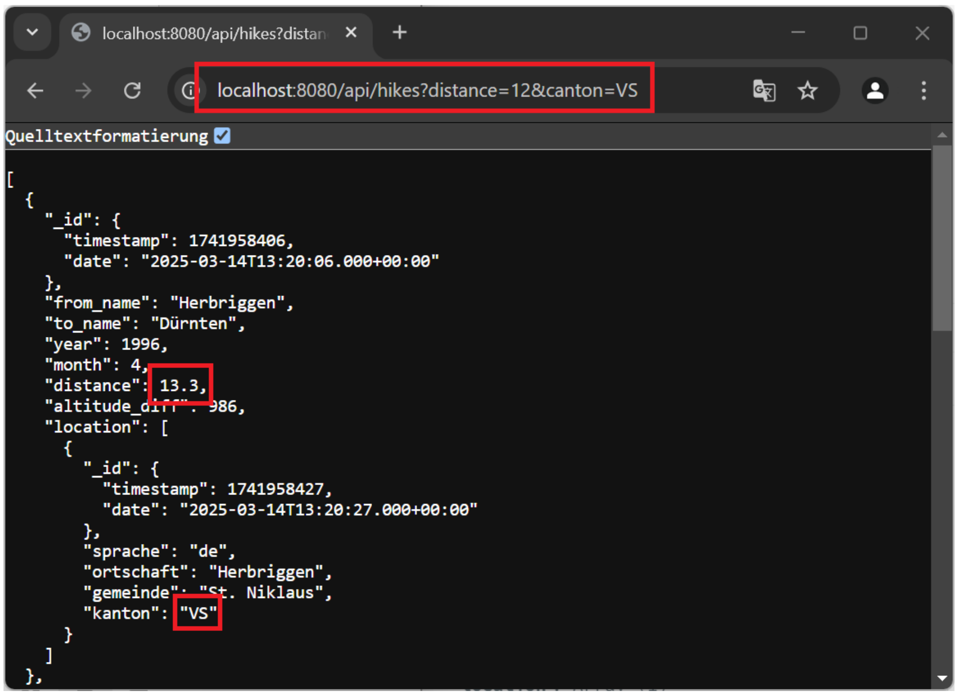
Quelltextformatierung ✓

```
[
  {
    "_id": {
      "timestamp": 1741958406,
      "date": "2025-03-14T13:20:06.000+00:00"
    },
    "from_name": "Vadura",
    "to_name": "Wisen SO",
    "year": 2000,
    "month": 3,
    "distance": 36,
    "altitude_diff": 402
  },
```

Optional Task

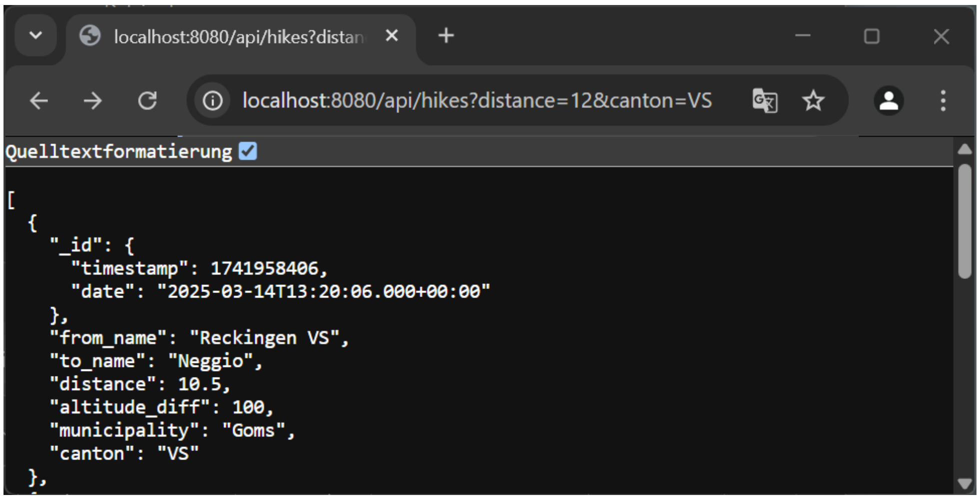
j) Design a new query that returns 3 random hikes filtered by length and by the canton where the hike starts. Note: the canton is not included in the hike collection. You must retrieve this information from the location collection. Tipp: You need a lookup stage for this.

k) Add «canton» as an optional parameter to the endpoint. Tipp: canton is not an int. Extend the endpoint with the new pipeline and verify that your backend filters the hikes correctly. Example: http://localhost:8080/api/hikes?distance=12&canton=VS


localhost:8080/api/hikes?distan
localhost:8080/api/hikes?distance=12&canton=VS
Quelltextformatierung

```
[
  {
    "_id": {
      "timestamp": 1741958406,
      "date": "2025-03-14T13:20:06.000+00:00"
    },
    "from_name": "Herbriggen",
    "to_name": "Dürnten",
    "year": 1996,
    "month": 4,
    "distance": 13.3,
    "altitude_diff": 986,
    "location": [
      {
        "_id": {
          "timestamp": 1741958427,
          "date": "2025-03-14T13:20:27.000+00:00"
        },
        "sprache": "de",
        "ortschaft": "Herbriggen",
        "gemeinde": "St. Niklaus",
        "kanton": "VS"
      }
    ]
  },
```

l) The current response contains many not required fields. Try to optimize the output as shown below. The field «municipality» contains the data from «gemeinde». Tipp: use $arrayElemAt to remove the arrays of municipality and canton.


localhost:8080/api/hikes?distan
localhost:8080/api/hikes?distance=12&canton=VS
Quelltextformatierung

```
[
  {
    "_id": {
      "timestamp": 1741958406,
      "date": "2025-03-14T13:20:06.000+00:00"
    },
    "from_name": "Reckingen VS",
    "to_name": "Neggio",
    "distance": 10.5,
    "altitude_diff": 100,
    "municipality": "Goms",
    "canton": "VS"
  },
```

m) Now you're ready to start with the frontend 😊.