



ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ
ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΙΔΙΚΕΥΣΗ: “ΠΡΟΗΓΜΕΝΑ ΠΛΗΡΟΦΟΡΙΑΚΑ
ΣΥΣΤΗΜΑΤΑ”

Διαχείριση Ναυτιλιακών Δεδομένων σε Μη Σχεσιακή Βάση Δεδομένων

Διαχείριση Δεδομένων για Σχεσιακές και Μη Σχεσιακές Βάσεις
Δεδομένων

Επιβλέπων Καθηγητής: Χ. Δουλκερίδης

Ιωάννα Κανδή (ME2136)

ΗΜΕΡΟΜΗΝΙΑ: Φεβρουάριος 2022

Πίνακας Περιεχομένων

Κατάλογος Εικόνων.....	1
Εισαγωγή.....	2
Τεχνολογίες Υλοποίησης.....	3
Οδηγίες Εγκατάστασης.....	3
1.Προσπέλαση Ναυτιλιακών Δεδομένων.....	4
2.Λειτουργίες Εφαρμογής	5
2.1 Query1: Return vessel details through sourcemmsi search	5
2.2 Query2: Return up to 5 vessels with speed greater than the one given by the user	7
2.3 Query3: Return a vessel's route through sourcemmsi	9
2.4 Query4: Locate vessels 8km away from coordinates specified by the user	11
3.Δημιουργία Ευρετηρίων (Indexes)	13
Συμπεράσματα.....	14

Κατάλογος Εικόνων

Εικόνα1-1: nari.marine data collection.....	4
Εικόνα 1-2: nari.marine data documents	4
Εικόνα 2.1.1: Φόρμα εκτέλεσης query1	5
Εικόνα 2.1.2: πίνακας Details με τα δεδομένα από το sourcemmsi	6
Εικόνα 2.1.3: κώδικας query1.....	6
Εικόνα 2.2.1: φόρμα εκτέλεσης query2.....	7
Εικόνα 2.2.2: αποτελέσματα εκτέλεσης query2. Εμφάνιση sourcemmsi και speedoverground.....	7
Εικόνα 2.2.3: κώδικας query2.....	8
Εικόνα 2.3.1: φόρμα εκτέλεσης query3.....	9
Εικόνα 2.3.2: αποτελέσματα εκτέλεσης query3.....	9
Εικόνα 2.3.3: κώδικας query3.....	10
Εικόνα 2.4.1: φόρμα εκτέλεσης query4.....	11
Εικόνα 2.4.2: αποτελέσματα εκτέλεσης query4.....	11
Εικόνα 2.4.3: κώδικας query4.....	12
Εικόνα 3.1: κώδικας ευρετηρίων (indexes)	13

Εισαγωγή

Η εκπόνηση εργασιών στα πλαίσια των μαθημάτων της Μεταπτυχιακής Εξειδίκευσης Σπουδών με τίτλο «Προηγμένα Πληροφοριακά Συστήματα» ήταν απαραίτητη για την μέγιστη δυνατή εξοικείωση των φοιτητών με το αντικείμενο του Προγράμματος. Η παρούσα εργασία αφορά το μάθημα χειμερινού εξαμήνου με τίτλο «Διαχείριση Δεδομένων για Σχεσιακές και Μη Σχεσιακές Βάσεις Δεδομένων», και πραγματεύεται το δεύτερο προτεινόμενο θέμα το οποίο αφορά τις Μη Σχεσιακές Βάσεις Δεδομένων. Ειδικότερα, γίνεται λόγος για διαχείριση δεδομένων ναυτιλίας μέσω ανάπτυξης εφαρμογής NoSQL ή MongoDB. Για αυτή την εργασία χρησιμοποιήθηκε η δεύτερη. Στο κείμενο που ακολουθεί αναπτύσσονται και επεξηγούνται οι λειτουργίες που εκτελεί η εφαρμογή «Nari Ships - Maritime Data», η δημιουργία ευρετηρίων (indexes) καθώς και η γραφική ανάπτυξη διεπαφής χρήστη (graphical user interface). Τέλος, παρατίθενται τα συμπεράσματα που προκύπτουν από την εκπόνηση της εργασίας αυτής.

Τεχνολογίες Υλοποίησης

Η υλοποίηση της εφαρμογής επιτεύχθηκε με τις παρακάτω τεχνολογίες:

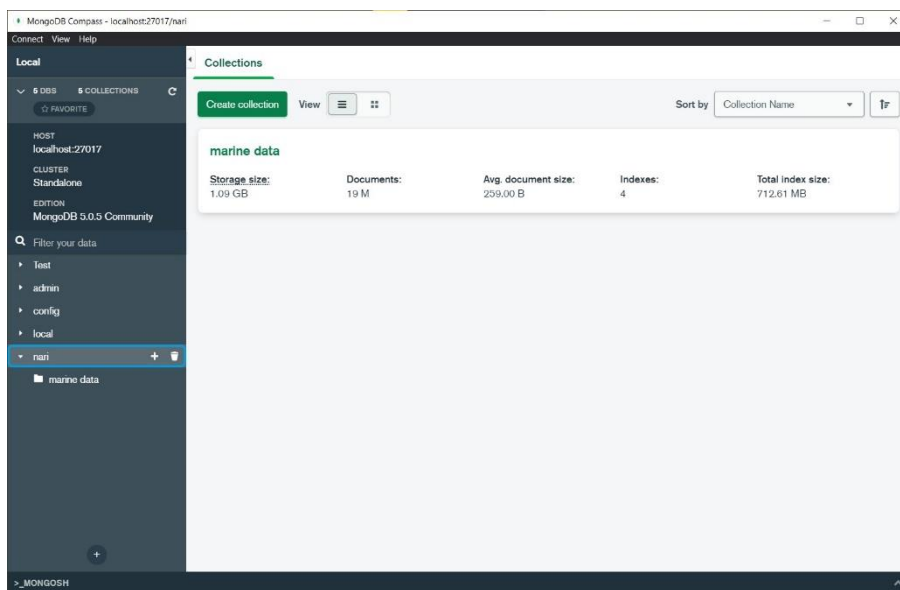
- Γλώσσα προγραμματισμού Python (έκδοση 3.9.7)
- Περιβάλλον Spyder IDE (έκδοση 5.1.5)
- MongoDB Compass (έκδοση 1.30.1)

Οδηγίες Εγκατάστασης

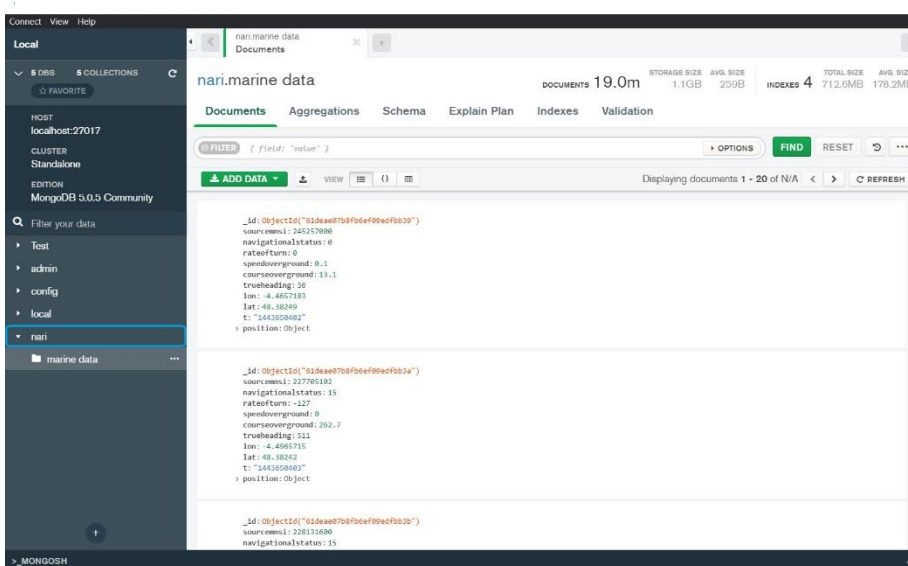
Για την επιτυχή εκτέλεση της εφαρμογής είναι σημαντική η εγκατάσταση της ανωτέρω έκδοσης της Python. Στη συνέχεια, μέσω της χρήσης terminal/prompt εντός του φακέλου στον οποίο βρίσκονται τα αρχεία της εφαρμογής, θα πρέπει να εγκατασταθούν οι απαραίτητες βιβλιοθήκες (requirements) μέσω της εντολής `'pip install -r requirements.txt'`. Επίσης, για να εκκινηθεί η εφαρμογή, θα πρέπει να πραγματοποιηθεί η εντολή `'python 3 pari_dynamic.py'`. Αφού εκτελεστούν οι παραπάνω ενέργειες, η εφαρμογή θα είναι διαθέσιμη στη διεύθυνση `http://localhost:5000`.

1. Προσέλαση Ναυτιλιακών Δεδομένων

Η εφαρμογή που αναπτύχθηκε αφορά την επεξεργασία ναυτιλιακών δεδομένων στη Μη Σχεσιακή Βάση Δεδομένων MongoDB. Για τις ανάγκες της εργασίας χρησιμοποιήθηκε το σύνολο δεδομένων «nari_dynamic» το οποίο εμπεριέχει δεδομένα κίνησης πλοίων. Προκειμένου να προσπελαστούν τα δεδομένα, δημιουργήθηκε η αντίστοιχη συλλογή (collection) «marine data» μέσω του MongoDB Compass.



Εικόνα1-1: nari.marine data collection



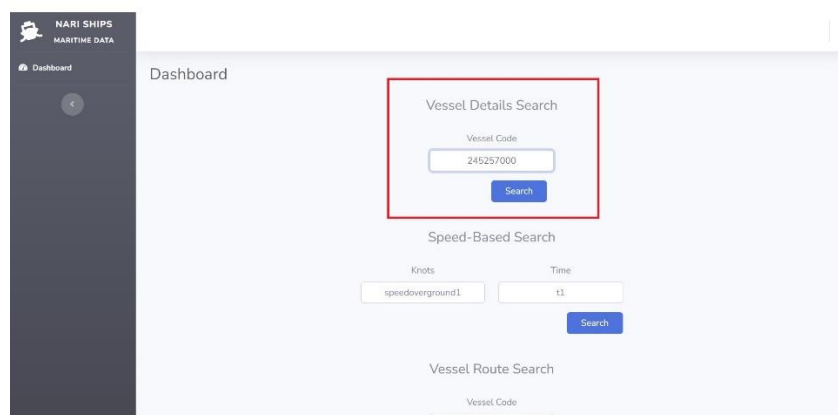
Εικόνα 1-2: nari.marine data documents

2.Λειτουργίες Εφαρμογής

Το παρόν κεφάλαιο περιλαμβάνει τις λειτουργίες (queries) που εκτελεί η εφαρμογή “Nari Ships - Maritime Data”. Πριν την ανάλυσή τους, πρέπει να επισημανθεί ότι για την ευκολότερη πλοήγηση της εφαρμογής από τον χρήστη, δημιουργήθηκε γραφική διεπαφή με αξιοποίηση των HTML, CSS, Bootstrap και JavaScript, ενώ για την εκτέλεση των εντολών στο back-end χρησιμοποιήθηκε η γλώσσα προγραμματισμού Python και η βιβλιοθήκη PyMongo για την πρόσβαση/σύνδεση στη MongoDB.

2.1 Query1: Return vessel details through sourcemmsi search

Η πρώτη λειτουργία της εφαρμογής είναι η επιστροφή των λεπτομερειών ενός πλοίου μέσω του κωδικού που πληκτρολογεί ο χρήστης. Αρχικά, εισάγει τον κωδικό του πλοίου¹ της επιλογής του, δηλαδή τη μεταβλητή sourcemmsi που θα αναζητηθεί στην εφαρμογή, στη φόρμα ‘Vessel Details Search’. Με την επιτυχή εκτέλεση του query αυτού, εμφανίζονται οι λεπτομέρειες του πλοίου στον πίνακα “Details”.



Εικόνα 2.1.1: Φόρμα εκτέλεσης query1

¹ Για την επίδειξη της εφαρμογής χρησιμοποιήθηκε το sourcemmsi: 245257000

Details								
VesselMmsi	NavigationalStatus	RateOfTurn	SpeedOverGround	CourseOverGround	TrueHeading	Longitude	Latitude	t
245257000	0	0	0.1	13.1	36	-4.4657183	48.38249	1443650402
245257000	0	0	0	13.1	35	-4.46572	48.382507	1443650413
245257000	0	0	0	13.1	35	-4.4657216	48.38252	1443650423
245257000	0	0	0.1	13.1	35	-4.4657283	48.382526	1443650433

Copyright © Ioanna 2022

Εικόνα 2.1.2: πίνακας Details με τα δεδομένα από το sourceemmsi

```
# endpoint for query1: return vessel details through sourceemmsi search
@app.route("/vesselSearch")
@cross_origin()
def vesselSearch():
    if request.method == 'GET':
        sourceemmsi = request.args.get('sourceemmsi')
        query = {'sourceemmsi':int(sourceemmsi)} #converts sourceemmsi input from str to int, so it can be identified by mongo
        result_cursor = marine_data.find(query, {"_id": 0}).limit(4) #because each vessel/sourceemmsi corresponds to multiple documents
                                                                    # the table filling has been limited to 4 sourceemmsi corresponding documents

        # convert cursor object to python list
        list_cur = list(result_cursor)
        return Response(json.dumps(list_cur), status=200, mimetype="application/json")
```

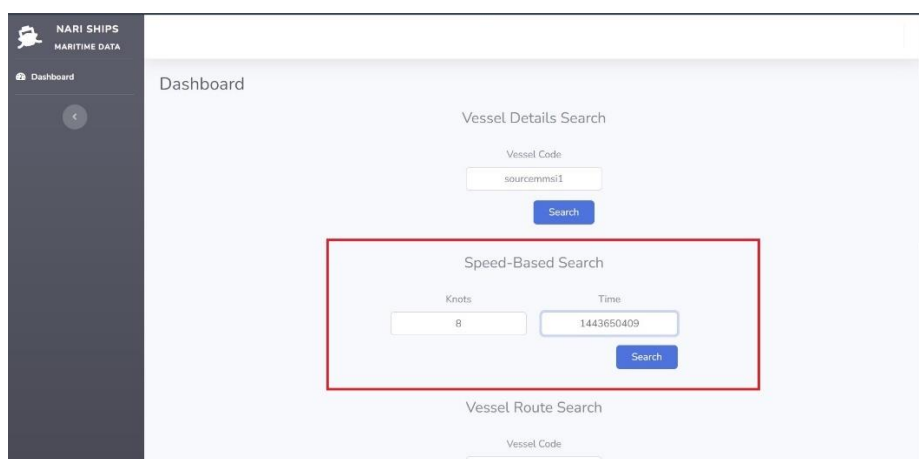
Εικόνα 2.1.3: κώδικας query1

Ο πίνακας Details της εικόνας 2.1.2 φαίνεται να απεικονίζει τον κωδικό του πλοίου και τα στοιχεία του σε παραπάνω από μία γραμμή, που ενδεχομένως να ήταν και το αναμενόμενο. Αυτό συμβαίνει διότι στη MongoDB, κάποια μεταβολή των εισαχθέντων δεδομένων αποθηκεύεται εκ νέου σε νέο document. Για παράδειγμα, το πλοίο που αναζητά ο χρήστης μπορεί να ταξιδεύει, επομένως σε κάθε διαφορετική χρονική στιγμή (t), τα longitude και latitude θα διαφέρουν. Επομένως, θα υπάρχουν πολλαπλά documents για έναν κωδικό/ sourceemmsi (έτσι όπως είναι δομημένη η συγκεκριμένη βάση δεδομένων). Για λόγους ευκολίας η εμφάνιση των documents περιορίστηκε στα τέσσερα (4). Ο χρόνος εκτέλεσης αυτής της λειτουργίας είναι 1s².

² Ο χρόνος αυτός είναι χωρίς την χρήση index

2.2 Query2: Return up to 5 vessels with speed greater than the one given by the user

Η επόμενη λειτουργία που δύναται να πραγματοποιήσει η πλατφόρμα «Nari Ships - Maritime Data», είναι η επιστροφή μέχρι και πέντε (5) πλοίων³ με ταχύτητα μεγαλύτερη από αυτή που εισήγαγε ο χρήστης. Σε αυτή την περίπτωση, καλείται να εισάγει την ταχύτητα (speedoverground) και τον χρόνο (t) που επιθυμεί μέσω της φόρμας 'Speed-Based Search' της εικόνας 2.2.1. Ο χρόνος αποτελεί απαραίτητη προϋπόθεση, καθώς ζητούνται πέντε πλοία μοναδικά και όχι πολλαπλά documents του ίδιου πλοίου. Άρα, αυτό που πραγματοποιεί το συγκεκριμένο query είναι η αναζήτηση των πλοίων με μεγαλύτερη ταχύτητα από την εισαχθείσα, σε μια συγκεκριμένη χρονική στιγμή που επίσης εισάγει ο χρήστης⁴. Η επιτυχής εμφάνιση των πέντε αυτών πλοίων είναι τυχαία. Ο χρόνος εκτέλεσης αυτής της λειτουργίας είναι 30s⁵.



Εικόνα 2.2.1: φόρμα εκτέλεσης query2

Details								
VesselMmsi	NavigationalStatus	RateOfTurn	SpeedOverGround	CourseOverGround	TrueHeading	Longitude	Latitude	t
228037600	-	-	9	-	-	-	-	-
227415000	-	-	11.4	-	-	-	-	-

Copyright © Ioanna Kandi me2136 2022

Εικόνα 2.2.2: αποτελέσματα εκτέλεσης query2. Εμφάνιση sourceemsi και speedoverground.

³ Περιορίστηκαν σε πέντε (5) για λόγους ευκολίας .

⁴ Για την επίδειξη της εφαρμογής χρησιμοποιήθηκε t=8 και speedoverground= 1443650409.

⁵ Ο χρόνος αυτός είναι χωρίς την χρήση index.


```

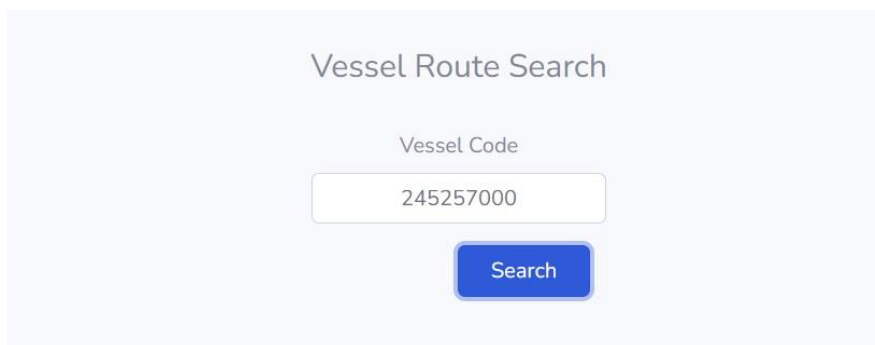
# endpoint for query2: return up to 5 vessels with speed greater than the one given by the user at a specific time, also provided by the user
@app.route("/knotSearch")
@cross_origin()
def knotSearch():
    if request.method == 'GET':
        speedoverground = request.args.get('speedoverground')
        t = request.args.get('t')
        query = { "speedoverground": { "$gt": int(speedoverground) }, "t":t }
        result_cursor = marine_data.find(query, {"_id": 0}).limit(5)
        # convert cursor object to python list
        list_cur = list(result_cursor)
        return Response(json.dumps(list_cur), status=200, mimetype="application/json")

```

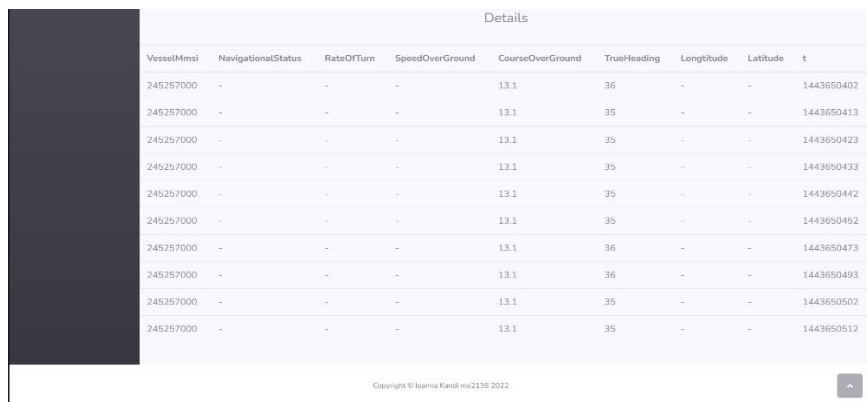
Εικόνα 2.2.3: κώδικας query2

2.3 Query3: Return a vessel's route through sourcemmsi

Η τρίτη διαδικασία που εκτελείται μέσω της συγκεκριμένης εφαρμογής είναι η εύρεση της πορείας που έχει χαράξει ένα πλοίο μέσω του sourcemmsi. Στην προκειμένη περίπτωση, ο χρήστης εισάγει ξανά το sourcemmsi της αρεσκείας του⁶ στη φόρμα 'Vessel Route Search', όπως φαίνεται στην εικόνα 2.3.1. Προκειμένου να εκτελεστεί ορθά το συγκεκριμένο ερώτημα, εκτός από την αναζήτηση του sourcemmsi στο 'marine data' collection πρέπει να ταξινομηθεί και ο χρόνος (t) κατά αύξουσα σειρά, εφόσον η πορεία του πλοίου διαφέρει από χρόνο σε χρόνο σε χρόνο. Η επιτυχής εκτέλεση της διαδικασίας αυτής, εμφανίζει τα στοιχεία⁷ που πλοίου σε κάθε διαφορετική χρονική στιγμή σε αύξουσα ταξινόμηση. Για λόγους ευκολίας εμφανίζονται μέχρι δέκα (10) διαφορετικά χρονικά στιγμιότυπα του πλοίου. Ο χρόνος εκτέλεσης είναι 5s.



Εικόνα 2.3.1: φόρμα εκτέλεσης query3



VesselMmsi	NavigationalStatus	RateOfTurn	SpeedOverGround	CourseOverGround	TrueHeading	Longitude	Latitude	t
245257000	-	-	-	13.1	36	-	-	1443650402
245257000	-	-	-	13.1	35	-	-	1443650413
245257000	-	-	-	13.1	35	-	-	1443650423
245257000	-	-	-	13.1	35	-	-	1443650433
245257000	-	-	-	13.1	35	-	-	1443650442
245257000	-	-	-	13.1	35	-	-	1443650452
245257000	-	-	-	13.1	36	-	-	1443650473
245257000	-	-	-	13.1	36	-	-	1443650493
245257000	-	-	-	13.1	35	-	-	1443650502
245257000	-	-	-	13.1	35	-	-	1443650512

Εικόνα 2.3.2: αποτελέσματα εκτέλεσης query3

⁶ Για τις ανάγκες του παραδείγματος χρησιμοποιήθηκε το sourcemmsi: 245257000

⁷ Sourcemmsi, courseoverground, trueheading, t

```

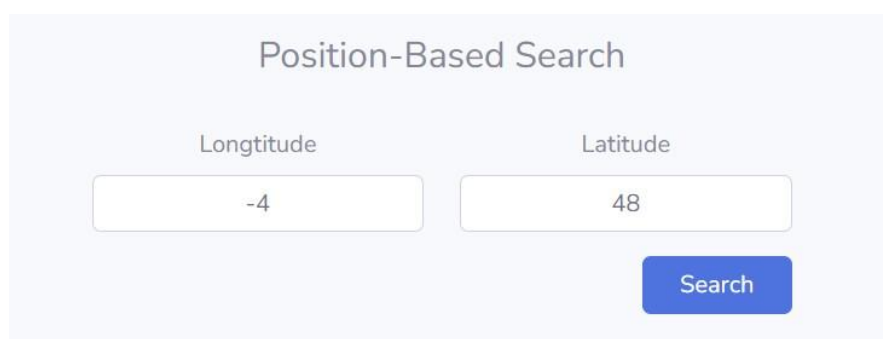
#endpoint for query3: return a vessel's route through sourceemmsi
#in order for this query to be executed successfully, the marine_data collection has to be skimmed through via sourceemmsi
#and sorted by time in ascending order.
#time (ascending) is necessary because a vessel's position varies from timestamp to timestamp.
#Again, because one sourceemmsi has multiple documents, they have been limited to 10.
@app.route("/getCourse")
@cross_origin()
def getCourse():
    if request.method == 'GET':
        sourceemmsi = request.args.get('sourceemmsi')
        query = { "sourceemmsi": int(sourceemmsi)}
        result_cursor = marine_data.find(query, {"_id": 0}).sort("t", 1).limit(10)
        # convert cursor object to python list
        list_cur = list(result_cursor)
        return Response(json.dumps(list_cur), status=200, mimetype="application/json")

```

Εικόνα 2.3.3: κώδικας query3

2.4 Query4: Locate vessels 8km away from coordinates specified by the user

Το τέταρτο και τελευταίο ερώτημα (query) που πραγματοποιείται μέσω της εφαρμογής “Nari Ships - Maritime Data” είναι ο εντοπισμός και κατ’ επέκταση η εμφάνιση των πλοίων που απέχουν 8km μακριά από τις συντεταγμένες που όρισε ο χρήστης⁸. Στη συγκεκριμένη λειτουργία, εισάγει τις τιμές γεωγραφικού μήκους (longitude) και γεωγραφικού πλάτους (latitude) που επιθυμεί στη φόρμα ‘Position-Based Search’ της εικόνας 2.4.1. Η επιτυχής εκτέλεση του ερωτήματος είναι η εμφάνιση των στοιχείων⁹ όλων των πλοίων που απέχουν 8km από τις επιθυμητές συντεταγμένες (εικόνα 2.4.2). Ο χρόνος εκτέλεσης είναι 44ms¹⁰.



Position-Based Search

Longitude: -4 Latitude: 48

Search

Εικόνα 2.4.1: φόρμα εκτέλεσης query4

Details								
VesselMmsi	NavigationalStatus	RateOfTurn	SpeedOverGround	CourseOverGround	TrueHeading	Longitude	Latitude	t
227116940	-	-	0	360	511	-	-	-
227003850	-	-	8.7	325	316	-	-	-
226338000	-	-	14.5	333.7	330	-	-	-
226338000	-	-	14.8	315.9	313	-	-	-
226338000	-	-	15.4	129.6	124	-	-	-
226338000	-	-	17.6	132.4	130	-	-	-

Copyright © Ioanna Kandi ms2136 2022

Εικόνα 2.4.2: αποτελέσματα εκτέλεσης query4

⁸ Για τις ανάγκες της εργασίας χρησιμοποιήθηκαν τα lon=-4 και lat=48

⁹ Επιστρέφονται τα πεδία sourcemmsi, speedoverground, courseoverground και trueheading.

¹⁰ Ο χρόνος αυτός είναι χωρίς τη χρήση index.

```

#endpoint for query4: locating vessels 8km away from given coordinates
@app.route("/vesselLocation",methods=['GET'])
@cross_origin()
def vesselLocation():
    if request.method == 'GET':
        #get the parameters from the request/user
        lat = request.args.get('lon')
        lon = request.args.get('lat')

        #find the sourceemsi of the vessels with distance 8km from given coordinates
        #merging lon and lat variables into one (coordinates) and setting maxDistance away from given coordinates as 8km
        #via geoNear aggregation pipeline.
        #returning chosen/necessary fields: sourceemsi, speedoverground, courseoverground, trueheading for each vessel corresponding to the user's input
        pipeline = [{"$geoNear": {"near": {"type":"Point", "coordinates": [int(lat), int(lon)]}, "distanceField": "dist.calculated", "maxDistance":8000,
                                "includeLocs": "dist.Location", "sphere": "true", "key": "position" }}
                    , {"$project": { "sourceemsi": 1, "speedoverground":1, "courseoverground":1,
                                "trueheading":1, "_id":0} }}]
        result_list = list(marine_data.aggregate(pipeline))

    return Response(json.dumps(result_list), status=200, mimetype="application/json")

```

Εικόνα 2.4.3: κώδικας query4

Κώλυμα της συγκεκριμένης διαδικασίας αποτέλεσε το γεγονός ότι οι συντεταγμένες που εισάγει ο χρήστης έπρεπε να ‘συγχωνευθούν’ σε μία , και από αυτήν να υπολογιστεί η μέγιστη απόσταση των 8 χιλιομέτρων, εφόσον ως μεταβλητές θέσης υπήρχαν μόνο τα longitude (lon) και latitude (lat) ξεχωριστά¹¹. Η εκτέλεση του συγκεκριμένου ερωτήματος (query) έγινε μέσω του ‘geoNear aggregation pipeline’ της βιβλιοθήκης PyMongo, όπως φαίνεται στο αντίστοιχο απόσπασμα κώδικα της εικόνας 2.4.3.

¹¹ Στο ‘marine data’ collection στη MongoDB. Η “συγχώνευση” των δύο μεταβλητών (lon, lat) γίνεται σε παρακάτω σημείο του κώδικα, στην ευρετηρίαση, το οποίο παρουσιάζεται στην αντίστοιχη ενότητα της εργασίας.

3.Δημιουργία Ευρετηρίων (Indexes)

Η ευρετηρίαση (indexing) είναι διαδικασία η οποία καθιστά αποδοτικότερη την εκτέλεση λειτουργιών (queries) που αφορούν την προσπέλαση αρχείων στη MongoDB. Πιο συγκεκριμένα, ένα ευρετήριο (index), αποθηκεύει και ψάχνει βάσει ενός συγκεκριμένου πεδίου(field) -ή συνόλου πεδίων- που χαρακτηρίζει το dataset μιας συλλογής από documents. Σε αντίθεση με την παραδοσιακή αναζήτηση σε όλη την έκταση του collection (collection scan) που ελέγχεται κάθε αρχείο (document) ξεχωριστά, η δημιουργία ευρετηρίων είναι αποδοτικότερη, καθώς είναι λιγότερο χρονοβόρο από την κλασική μέθοδο αναζήτησης.

Για τις ανάγκες της εργασίας, δημιουργήθηκαν τρία (3) ευρετήρια (indexes), όπως φαίνονται στην εικόνα 3.1. Το πρώτο ευρετήριο (2dsphere index) αφορά τις συντεταγμένες που αναζητούνται στο query4. Το δεύτερο στη σειρά, ψάχνει το dataset βάσει του κωδικού του πλοίου, δηλαδή του sourceemsi. Το τρίτο και τελευταίο ευρετήριο σχετίζεται με την ταχύτητα του πλοίου (speedoverground). Η εμφανής διαφορά στους χρόνους εκτέλεσης γίνεται εμφανής στον αντίστοιχο πίνακα 1.

```
#endpoint for creating indexes to minimize query execution time
@app.route("/generateIndex",methods=['GET'])
@cross_origin()
def generateIndex():
    if request.method == 'GET':

        #1) 2dsphere index for coordinates in query4
        marine_data.updateMany({},[{"$set":{"position":{"type":"Point","coordinates":["$lon","$lat"]}}}] )
        #create the coordinates index that contains both longitude and latitude (it will be used for
        #searching the collection based on coordinates)
        marine_data.create_index([("position", GEOD2D)])

        #2) sourceemsi index

        marine_data.create_index([('sourceemsi',pymongo.ASCENDING)],
                                name='sourceemsi')

        #3) speedoverground index
        marine_data.create_index([('speedoverground',pymongo.ASCENDING)],
                                name='speedoverground')

    return Response("Message:Indexes genetared.", status=200, mimetype="application/json")
```

Εικόνα 3.1: κώδικας ευρετηρίων (indexes)

Πίνακας 1: Χρόνοι εκτέλεσης χωρίς ευρετήριο και με ευρετήριο

Queries	T without index	T with index
Query1	1s	39ms
Query2 ¹²	30s	-
Query3	5s	70ms
Query4	44ms	25ms

¹² Το query2 δεν χρησιμοποιεί κάποιο index , για αυτό και δεν υπάρχει σύγκριση χρόνων εκτέλεσης.

Συμπεράσματα

Συνοψίζοντας, η ανάπτυξη εφαρμογής διαχείρισης ναυτιλιακών δεδομένων 'Nari Ships-Maritime Data' ήταν απαραίτητη για να επιτευχθεί μια πρώτη επαφή με τη διαχείριση δεδομένων σε Μη Σχεσιακή βάση δεδομένων. Η προαναφερθείσα εφαρμογή εκτελεί συγκεκριμένες διεργασίες και διαθέτει γραφική διεπαφή φιλική προς τον χρήστη έτσι ώστε να είναι πιο αρμονική η πλοήγηση, η εισαγωγή και η εμφάνιση των εκάστοτε επιθυμητών στοιχείων. Ωστόσο, υπάρχουν περιθώρια βελτίωσης. Ειδικότερα, η εφαρμογή αυτή θα μπορούσε, αρχικά, να επεξεργάζεται και να εκτελεί λειτουργίες αυξημένης πολυπλοκότητας. Επίσης, καλό θα ήταν να γίνεται επεξεργασία μεγαλύτερου όγκου δεδομένων, το οποίο θα μπορούσε να γίνει εφικτό μέσω της χρήσης περισσότερων συλλογών/collections (αυτή τη στιγμή αξιοποιείται μόνο μία). Τέλος, σημαντικό θα ήταν γίνει έλεγχος της εφαρμογής σε clusters μέσα από πλήθος ηλεκτρονικών υπολογιστών, κάτι το οποίο δεν είναι υλοποιήσιμο την περίοδο υλοποίησης της εφαρμογής.

