Web Information System and Applications

Project Report from

Stavros Droutsas, stavrosdro@gmail.com, cs2180026

Alexandros Fotios Ntogramatzis, ntogram@di.uoa.gr, cs2180010

Title: “RIAQ: Report Incidents and Ask Queries”

Περιγραφή εφαρμογής:

Η εφαρμογή εξυπηρετεί τις ακόλουθες ανάγκες:

- Κατάθεση report για συμβάντα στην πόλη:
  - Abandoned Vehicles
  - Street Lights Out
  - Damaged Garbage Carts
  - Graffiti Removal
  - Sanitation Code Violation
  - Tree Debris
  - Crimes

- Κατάθεση report για τροχαία ατυχήματα:
  - Traffic Crashes

- Άλλες χρήσιμες πληροφορίες για την πόλη
  - Land Inventory:αναφέρει πληροφορίες για τις εκτάσεις που είναι διαθέσιμες προς πώληση ή έχουν ήδη πουληθεί
  - Affordable Houses:πληροφορίες για τα διαθέσιμα προς πώληση σπίτια
  - Food Inspections:Πληροφορίες για εστιατόρια
  - Bike Racks: πληροφορίες για τα διαθέσιμα ποδήλατα

- Δεδομένα για κίνηση στους δρόμους
  - Average Daily Traffic

- Εκτέλεση ερωτημάτων για προβολή όλων των ανωτέρω ως προς:
  - Οδό
  - Zip-Code

- Chat για τους χρήστες με δωμάτιο ανάλογα με το θέμα συζήτησης.

Χρησιμότητα εφαρμογής:

Πολλοί άνθρωποι επισκέπτονται άλλες πόλεις είτε για δουλειά είτε για λόγους αναψυχής. Παλαιότερα, οι άνθρωποι για να προετοιμαστούν για την πρώτη επίσκεψη τους σε μια άλλη πόλη, αγόραζαν και διάβαζαν ταξιδιωτικούς οδηγούς & φυλλάδια από ταξιδιωτικά γραφεία με χρηστικές πληροφορίες για την πόλη που θα επισκεπτόταν. Σήμερα θα σπαταλήσουν πάλι πολλές ώρες αλλά αντί να διαβάζουν σχετικά βιβλία, θα ψάχνουν επίμονα σε διάφορες σελίδες του web σχετικές με την πόλη που θα επισκεφτούν και μάλιστα αυτό πολλές φορές συμβαίνει και όταν έχουν ήδη επισκεφτεί την πόλη και βρίσκονται στους δρόμους της. Τελικά μετά από αρκετή προσπάθεια καταφέρνουν να μάθουν τις πληροφορίες που θέλουν, Θα ήταν γενικά πολύ χρήσιμο να υπήρχε μια εφαρμογή που θα είχε μαζέψει όλες αυτές της πληροφορίες σε μια μεγάλη βάση δεδομένων για πολλά διαφορετικά μέρη στον κόσμο και μέσα από ερωτήματα σε αυτή οι πολίτες θα έβρισκαν μαζεμένες όλες τις χρήσιμες πληροφορίες που είτε έπρεπε να τις ψάξουν στις σελίδες ενός ταξιδιωτικού οδηγού ή σε διάφορα σημεία στο διαδίκτυο για να τις βρουν.

Στα πλαίσια αυτής εργασίας επιχειρήθηκε κάτι τέτοιο αλλά σε μικρότερη κλίμακα. Συγκεκριμένα μαζεύτηκαν τα δεδομένα από κάποια datasets για την πόλη του Chicago (υπάρχουν link για τα dataset στο τέλος του Report) και μέσω ερωτημάτων, οι χρήστες μπορούν να μάθουν κάποιες πληροφορίες για ακίνητα, καταστήματα και συμβάντα στην πόλη του Chicago. Σκεφτήκαμε ότι για να εμπλουτίζεται με δεδομένα και πληροφορίες η βάση δεδομένων της εφαρμογής, να δίνεται στους χρήστες η δυνατότητα να κάνουν εισαγωγή συμβάντων μέσω φιλικών προς τον χρήστη φορμών.

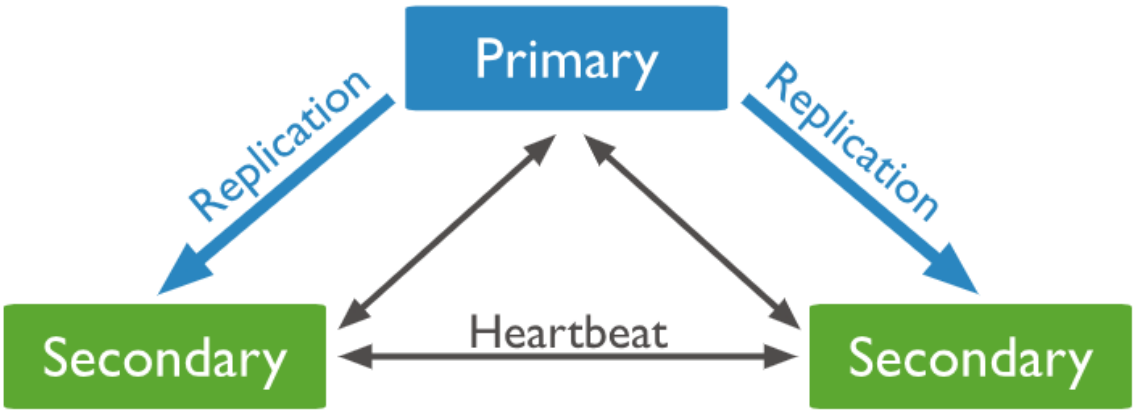
Τέλος αξίζει να σημειωθεί ότι μια τέτοια εφαρμογή θα ήταν πολύ χρήσιμη και για τους πολίτες της πόλης του Chicago.

Σχεδιασμός εφαρμογής

Για βάση χρησιμοποιήσαμε MongoDB καθώς δεν είναι απαραίτητη η εκτέλεση transactions για τη λειτουργία της εφαρμογής, ενώ παράλληλα έχουμε μεγάλο κέρδος στην απόδοση καθώς εκμεταλλευόμαστε replication και sharding και προσφέρει μεγάλη ευελιξία

Replication

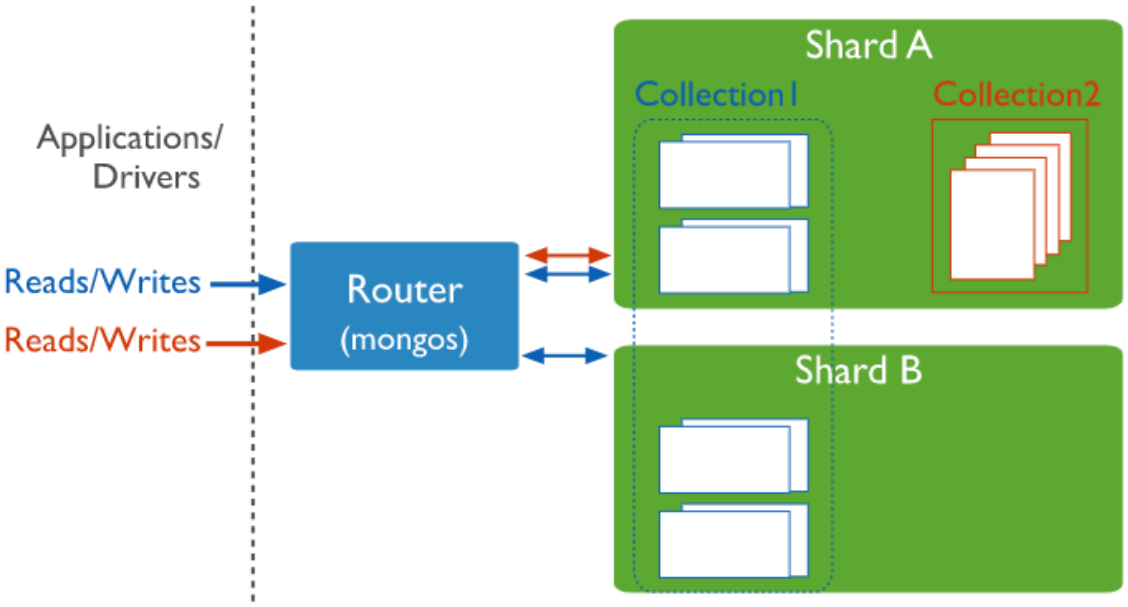
Αρχικά χρησιμοποιήσαμε 3 replications της βάσης άρα είχαμε 3 instances (ένα primary, 2 secondary αποφασίζονται με ψηφοφορία μεταξύ τους), ώστε αν κάποιο πέσει να συνεχίζουν να εκτελούνται queries δίχως downtime. Στον τομέα του scaling το κέρδος είναι ότι τα reading operations χρησιμοποιούν όλα τα replicas (που αποτελούν και το περισσότερο workload των εφαρμογών) και παράλληλα έχουμε backup των δεδομένων σε περιπτώσεις που συμβούν αποτυχίες στο σύστημα .Έτσι το σύστημα έχει μεγαλύτερο reliability και fault-tolerance.Το παρακάτω σχήμα αναπαριστά τη βασική λογική λειτουργίας του replication:



Το πρώτο κέρδος στην απόδοση έρχεται καθώς για τα reading operations μπορούν να συμβάλλουν και τα 3 αντίγραφα.

Sharding

Λόγω του μεγάλου όγκου δεδομένων αποφασίστηκε η βάση να γίνει distributed. Ο σκοπός είναι το κύριο collection να μοιραστεί σε διαφορετικά μηχανήματα. Με αυτόν τον τρόπο έχουμε πολλές παράλληλες διαδικασίες ώστε να έχουμε κέρδος στα writing operations και τα reading να επιταχυνθούν ακόμη περισσότερο. Στο παρακάτω σχήμα παρουσιάζεται ο τρόπος λειτουργίας του sharding .Ένας router υπάρχει για να κατευθύνει τα ερωτήματα στο αντίστοιχο shard.



Η τελική υλοποίηση έχει 3 shards και το καθένα αποτελείται από 3 replica sets. Σχετικά με τον καταμερισμό των εγγραφών χρησιμοποιήσαμε ένα hash function πάνω στο zip-code των δεδομένων ώστε να είναι χωρισμένα ισάξια στο κάθε shard. Μια τέτοια προσέγγιση είναι βολική για ερωτήματα που αφορούν συγκεκριμένα zip-codes και γενικά για ερωτήματα που ψάχνουμε για στοιχεία που είναι σε μικρές αποστάσεις μεταξύ τους και συνεπώς το πιο πιθανό είναι να έχουν το ίδιο zipcode. Με αυτό τον τρόπο εκμεταλλευόμαστε καλύτερα το locality της πληροφορίας.

Schema

Υπάρχουν 2 collections ένα για τα Incidents και ένα για τους Users. Οι εγγραφές στο Incidents έχουν κάποια κοινά πεδία και κάποια επιπλέον πεδία που διαφέρουν ανάλογα με τον τύπο του συμβάντος

Τα κοινά πεδία των εγγραφών είναι τα ακόλουθα:

```
private String id;
private String date;
private String type;
private Integer zipcode;
private String streetaddress;
private Integer ward;
private Double latitude;
private Double longitude;
```

Affordable houses

```
private String communityname;
private Integer sqfeet;
private String phonenumber;
private String managementcompany;
```

Bike Racks

```
private String communityname;
```

Crashes

```
private String time;
private String weathercondition;
private String lightingcondition;
private String crashtype;
```

Crimes

```
private String primarytype;
private String locationdescription;
```

Daily Traffic

```
private Integer vehiclenumber;
private String vehicledirection;
```

Land Inventory

```
private String propertystatus;
private Integer sqfeet;
```

Restaurants

```
private String inspectiontype;
private String results;
private String violations;
private String name;
private String facilitytype;
```

Users

```
@Id
private String id;
@Indexed(unique = true, direction = IndexDirection.DESCENDING,
dropDups = true)
private String email;
private String password;
@Transient
private String passwordConfirm;
private String fullname;
private boolean enabled;
@DBRef
private Set<Role> roles;
```

Indexes

Για ταχύτερη εκτέλεση ερωτημάτων χρησιμοποιήθηκαν τα ακόλουθα ευρετήρια

- Διπλό ευρετήριο σε latitude – longitude (συνεχής χρήση για bounding-box σύμφωνα με το δρόμο που εισάγει ο χρήστης).
- Τριπλό ευρετήριο σε type – _class – sqfeet (type of incidents και _class για τον τύπο των ερωτημάτων και την αντίστοιχη Java κλάση και sqfeet για τις περιπτώσεις εύρεσης κατοικίας.

Chat

Το chat βασίζεται σε ένα πρωτόκολλο WebSocket με το οποίο δημιουργείται ένα κανάλι αμφίδρομης επικοινωνίας μεταξύ ενός server και ενός client. Αρχικά δημιουργεί μια HTTP connection την οποία κάνει upgrade μέσω ενός αντίστοιχου header σε μια αμφίδρομη σύνδεση WebSocket. Κατά το configuration του WebSocket (WebSocket endpoint, message broker) γίνεται εκκίνηση ενός server κάνοντας register το endpoint στους clients. Χρησιμοποιείται το πρωτόκολλο ανταλλαγής μηνυμάτων STOMP που ορίζει τη μορφή και τους κανόνες ανταλλαγής δεδομένων και είναι απαραίτητο για τον ορισμό λειτουργειών που δεν ορίζει το απλό WebSocket πρωτόκολλο όπως την ανταλλαγή μηνυμάτων μεταξύ χρηστών που βρίσκονται σε συγκεκριμένο topic. Ο message broker είναι ένας in-memory και υπάρχει για να δρομολογεί τα μηνύματα στους clients που είναι εκείνη τη στιγμή ενεργοί στο συγκεκριμένο θέμα. Κάθε χρήστης μπορεί να βλέπει ένα topic κάθε στιγμή και συνεπώς αποχωρεί από ένα δωμάτιο για να εισέλθει σε ένα άλλο.

Αναζήτηση incident με βάση τη διεύθυνση

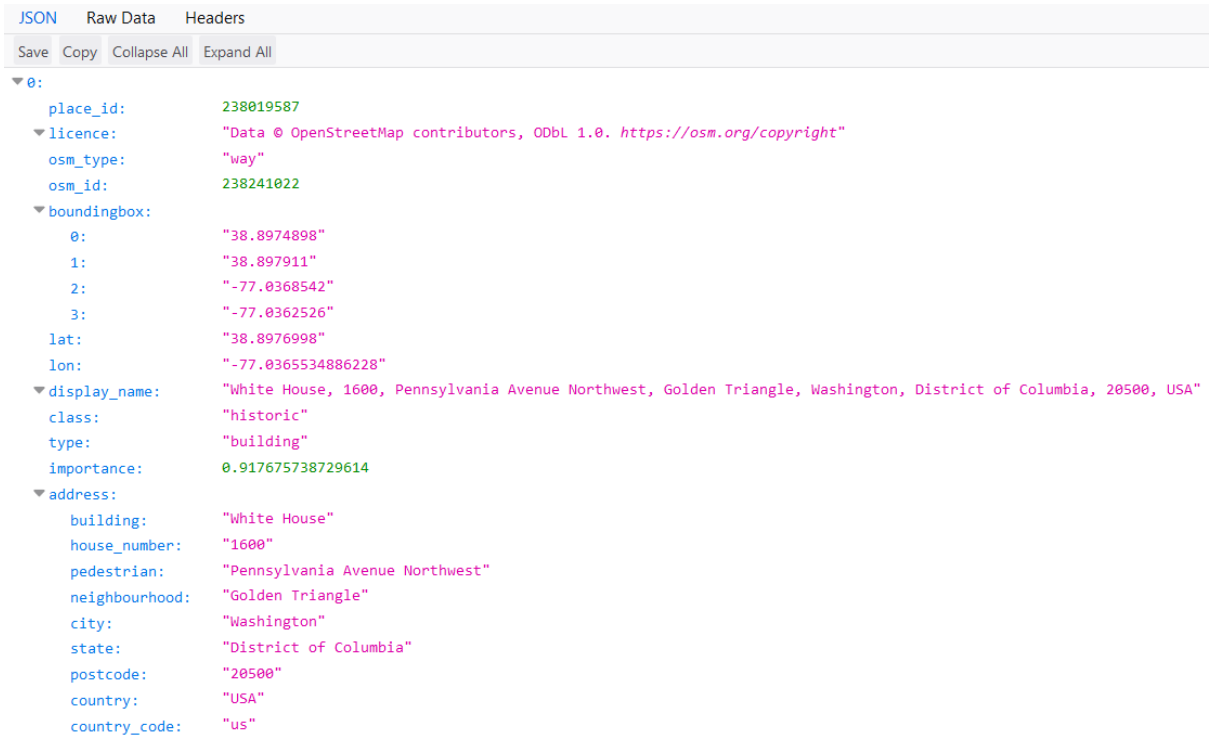
Για την αναζήτηση με βάση τη διεύθυνση πρέπει να γνωρίζουμε τις συντεταγμένες (lat – lon) που αντιστοιχούν στη διεύθυνση που εισάγει ο χρήστης. Η πιο απλή επιλογή θα ήταν να παίρνουμε τις πληροφορίες από την ίδια την βάση αλλά εδώ προτιμήθηκε να χρησιμοποιηθούν οι πληροφορίες από τη σελίδα

https://nominatim.openstreetmap.org/

Πρακτικά με κώδικα σε java γράφονται ερωτήματα τα οποία υποβάλλονται με post request προς την σελίδα αυτή π.χ.

https://nominatim.openstreetmap.org/search?q=%22The%20White%20House,%20Washington%20DC%22&format=json&addressdetails=1

και ζητείται από τη σελίδα να επιστρέψει κάποια αποτελέσματα αν υπάρχουν για τη ζητούμενη διεύθυνση ή περιοχή στο παράδειγμα εδώ αναζητείται το σημείο που είναι ο Λευκός Οίκος στην Ουάσιγκτον και ζητείται τα αποτελέσματα είναι JSON FORMAT.



Υπάρχουν περιπτώσεις που επιστρέφονται πολλά αποτελέσματα για τη διεύθυνση που ζητάει ένας χρήστης(ένα array από json objects) γεγονός που συμβαίνει σε περιπτώσεις που ένας δρόμος ή διεύθυνση που ζητάει κάποιος χρήστης υπάρχει σε περισσότερες περιοχές μιας πόλης ή σε περισσότερες πόλεις(στα πλαίσια αυτής της εφαρμογής μας ενδιαφέρει το πρώτο) τότε εμφανίζεται μια λίστα με το attribute display_name για κάθε json object του array και ο χρήστης θα καλείται να επιλέξει τη διεύθυνση που θέλει. Αφού επιλέξει, προσδιορίζονται από το αντίστοιχο json object:

- Οι γεωγραφικές συντεταγμένες του σημείου που βρίσκεται.
- Οι γεωγραφικές συντεταγμένες ενός bounding box γύρω από το ζητούμενο σημείο.

Σε περίπτωση που η εφαρμογή μας χρησιμοποιείται από πολλούς χρήστες και ένα μεγάλο μέρος τους δίνει μια διεύθυνση για να ψάξει διάφορα συμβάντα στη γύρω περιοχή, είναι ιδιαίτερο επιβαρυντικό για τον server να στέλνει συνεχώς http requests στο url που αναφέρεται παραπάνω και να περιμένει για να γίνει αναζήτηση για να βρεθούν οι συντεταγμένες της διεύθυνσης αυτής. Επιπλέον η εξυπηρέτηση του χρήστη γίνεται με κάποιο latency .Αυτό το πρόβλημα , αντιμετωπίστηκε αξιοποιώντας την δυνατότητα που μας δίνει το Springboot Framework για cache. Έτσι λοιπόν τροποποιήθηκε κατάλληλα το πρόγραμμα και προστέθηκε μια cache στη διαδικασία αναζήτησης συμβάντων με βάση την

διεύθυνση. Έτσι την πρώτη φορά που ο χρήστης θα ψάξει για Incidents σε μια διεύθυνση π.χ 5629 N KEDVALE AVE, θα γίνει το http request στην παραπάνω διεύθυνση και αυτή θα επιστρέψει ένα JSON Object σαν απάντηση από το οποίο θα πάρουμε τις συντεταγμένες τις διεύθυνσης και στη συνέχεια θα εκτελεστεί το ερώτημα. Στη συνέχεια αν ο χρήστης ψάξει για την περιοχή της ίδιας διεύθυνσης για Crimes, δεν θα σταλεί http request στην παραπάνω διεύθυνση αλλά θα παρθούν οι συντεταγμένες από το αποθηκευμένο json object στην cache.

Επιπλέον προστέθηκε η δυνατότητα οι χρήστες να μπορούν να δουν τις περιοχές με βάση τα zipcode όπου συμβαίνουν τα περισσότερα συμβάντα.

Τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής

Το back end της εφαρμογής έχει υλοποιηθεί σε spring-boot με χρήση της Java. Για το authentication/login έγινε χρήση sprint-security. Το dependency της MongoDB χρησιμοποιήθηκε για εισαγωγή εγραφών και εκτέλεση ερωτημάτων. Για το front end χρησιμοποιήθηκε JavaScript, HTML, Thymeleaf, CSS (Bootstrap).

MongoDB - configuration

Replicas

```
start mongod --replSet testrep --logpath \data\rs1\logs\1.log
--dbpath \data\rs1 --port 27018 --smallfiles --oplogSize 64

start mongod --replSet testrep --logpath \data\rs2\logs\2.log
--dbpath \data\rs2 --port 27019 --smallfiles --oplogSize 64

start mongod --replSet testrep --logpath \data\rs3\logs\3.log
--dbpath \data\rs3 --port 27020 --smallfiles --oplogSize 64

mongo --port 27018

config =
{_id:"testrep",members:[{_id:0,host:"localhost:27018"},{_id:1,
host:"localhost:27019"},{_id:2,host:"localhost:27020"}]};

rs.initiate(config)
```

Με τον παραπάνω κώδικα σε windows σηκώνουμε 3 instances της Mongo στα ports 27018, 27019, 27020

Μπορούμε να συνδεθούμε με την εντολή:

```
mongo --port 27018
mongo --port 27019
mongo --port 27020
```

ένα εκ των ανωτέρω είναι primary. Για να επιτρέψουμε εκτέλεση queries σε secondary εκτελούμε:

```
rs.slaveOk()
```

Τέλος για τον έλεγχο της συστοιχίας χρησιμοποιούμε την εντολή:

```
rs.status()
```

Sharding

Δημιουργούμε 3 shards που αποτελούνται από 3 replicas το καθένα:

```
mkdir \data\shard0\rs0 \data\shard0\rs1 \data\shard0\rs2

start mongod --replSet s0 --logpath "s0-r0.log" --dbpath \data\shard0\rs0 -
-port 37017 --shardsvr --smallfiles

start mongod --replSet s0 --logpath "s0-r1.log" --dbpath \data\shard0\rs1 -
-port 37018 --shardsvr --smallfiles

start mongod --replSet s0 --logpath "s0-r2.log" --dbpath \data\shard0\rs2 -
-port 37019 --shardsvr --smallfiles

mongo --port 37017

config = { _id: "s0", members:[{ _id : 0, host : "localhost:37017" },{ _id
: 1, host : "localhost:37018" },{ _id : 2, host : "localhost:37019" }]};

rs.initiate(config)


mkdir \data\shard1\rs0 \data\shard1\rs1 \data\shard1\rs2

start mongod --replSet s1 --logpath "s1-r0.log" --dbpath \data\shard1\rs0 -
-port 47017 --shardsvr --smallfiles

start mongod --replSet s1 --logpath "s1-r1.log" --dbpath \data\shard1\rs1 -
-port 47018 --shardsvr --smallfiles

start mongod --replSet s1 --logpath "s1-r2.log" --dbpath \data\shard1\rs2 -
-port 47019 --shardsvr --smallfiles

mongo --port 47017

config = { _id: "s1", members:[{ _id : 0, host : "localhost:47017" },{ _id
: 1, host : "localhost:47018" },{ _id : 2, host : "localhost:47019" }]};

rs.initiate(config)


mkdir \data\shard2\rs0 \data\shard2\rs1 \data\shard2\rs2

start mongod --replSet s2 --logpath "s2-r0.log" --dbpath \data\shard2\rs0 -
-port 57017 --shardsvr --smallfiles

start mongod --replSet s2 --logpath "s2-r1.log" --dbpath \data\shard2\rs1 -
-port 57018 --shardsvr --smallfiles

start mongod --replSet s2 --logpath "s2-r2.log" --dbpath \data\shard2\rs2 -
-port 57019 --shardsvr --smallfiles

mongo --port 57017

config = { _id: "s2", members:[{ _id : 0, host : "localhost:57017" },{ _id
: 1, host : "localhost:57018" },{ _id : 2, host : "localhost:57019" }]};

rs.initiate(config)
```

Με τις ακόλουθες εντολές δημιουργούμε το configuration server με τις ρυθμίσεις για να λειτουργήσει ως κατανεμημένη η βάση, που διαθέτει και αυτός 3 replicas:

```
mkdir \data\config\config-a \data\config\config-b \data\config\config-c

start mongod --replSet cs --logpath "cfg-a.log" --dbpath
\data\config\config-a --port 57040 --configsvr --smallfiles

start mongod --replSet cs --logpath "cfg-b.log" --dbpath
\data\config\config-b --port 57041 --configsvr --smallfiles

start mongod --replSet cs --logpath "cfg-c.log" --dbpath
\data\config\config-c --port 57042 --configsvr --smallfiles

mongo --port 57040

config = { _id: "cs", members:[{ _id : 0, host : "localhost:57040" },{ _id
: 1, host : "localhost:57041" },{ _id : 2, host : "localhost:57042" }]};

rs.initiate(config)
```

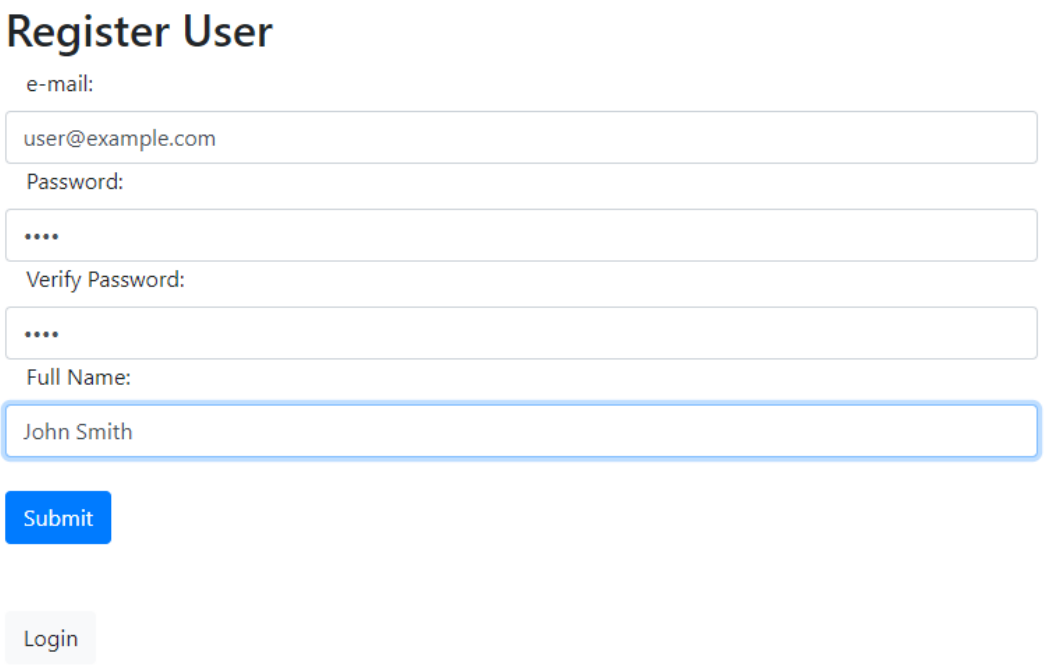
Τέλος ο router που προωθεί το ερώτημα στο αντίστοιχο shard:

```
mongos --logpath "mongos-1.log" --port 30000 --configdb
cs/localhost:57040,localhost:57041,localhost:57042
mongo --port 30000

db.adminCommand( { addshard : "s0/"+"localhost:37017" } );

db.adminCommand( { addshard : "s1/"+"localhost:47017" } );

db.adminCommand( { addshard : "s2/"+"localhost:57017" } );

db.adminCommand({enableSharding: "Incidents"});

sh.shardCollection( "Incidents.Incidents", { zipcode : "hashed" } )
```

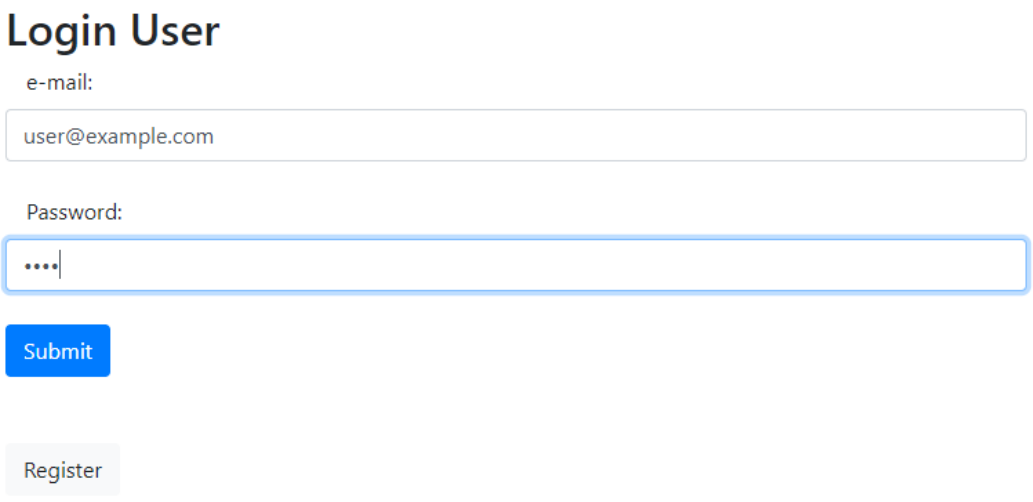
Η τελευταία γραμμή καθορίζει την κατανομή των εγγραφών. Στον παραπάνω κώδικα το πεδίο zipcode γίνεται hashed. Εναλλακτικά η κατανομή μπορεί να γίνει σύμφωνα με το εύρος του κλειδιού που επιλέχθηκε ή ακόμη και να επιλεχθούν συγκεκριμένα κλειδιά για το εκάστοτε shard.

Μερικά screenshots από την εφαρμογή:

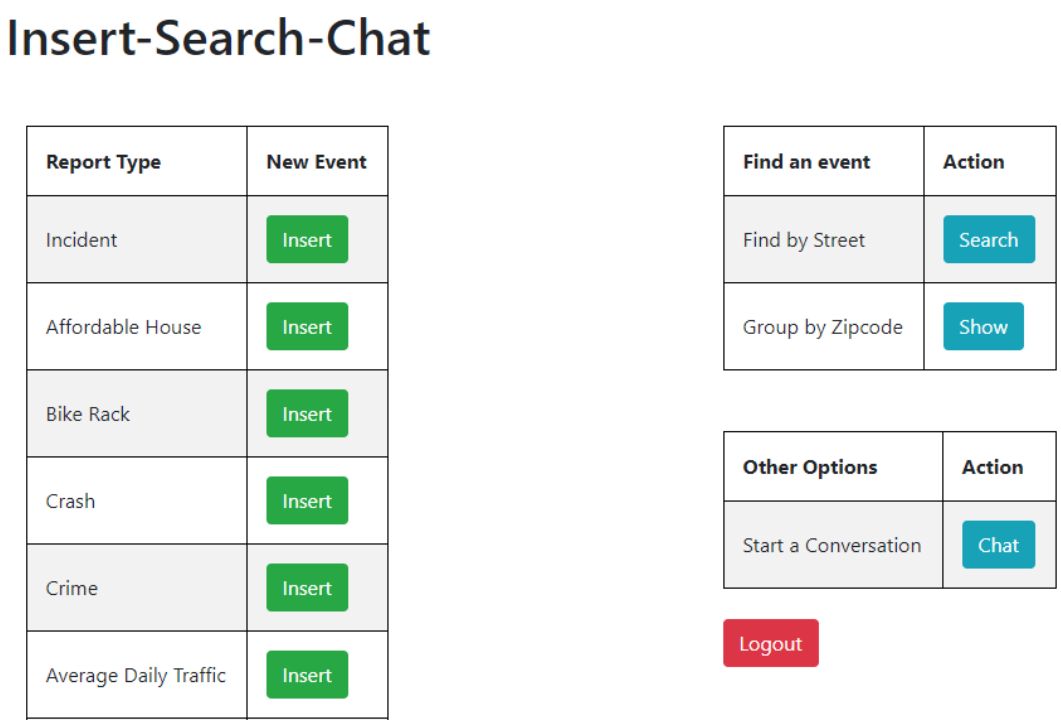
Register new user



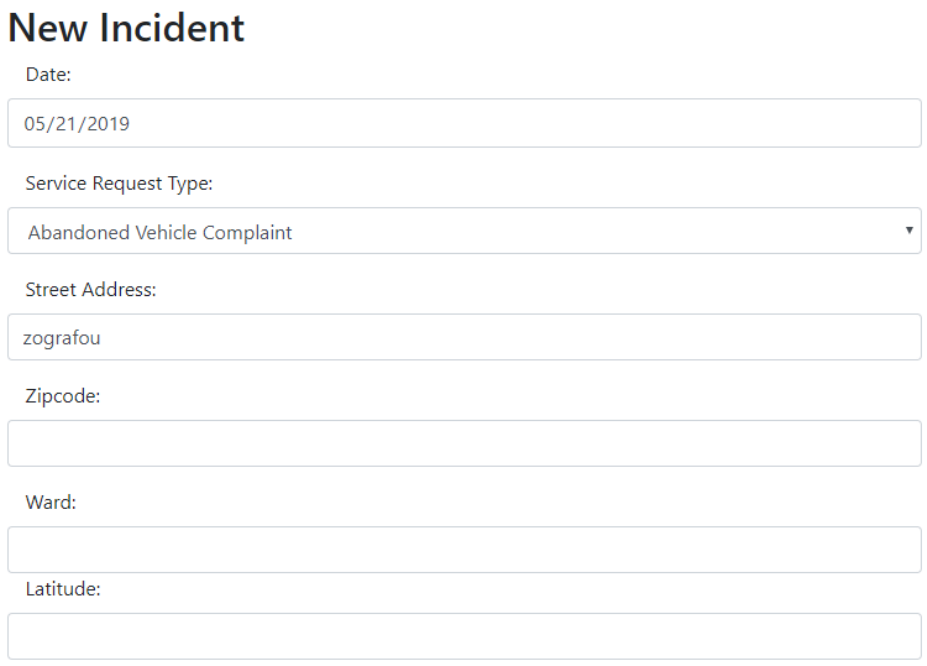
Login



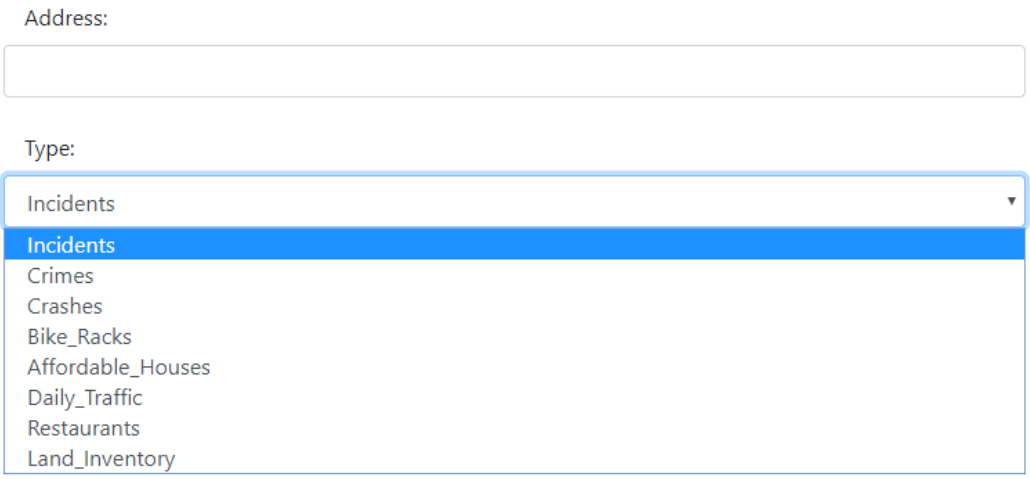
Home page

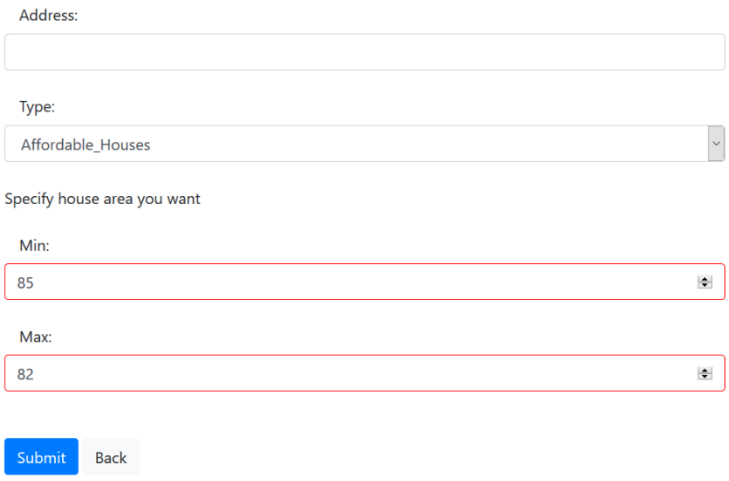


Insert an incident



Search with street address





Available Address:

Results

for region of following address

Address: 1008, East 41st Place, Oakland, Chicago, Cook County, Illinois, 60653, USA

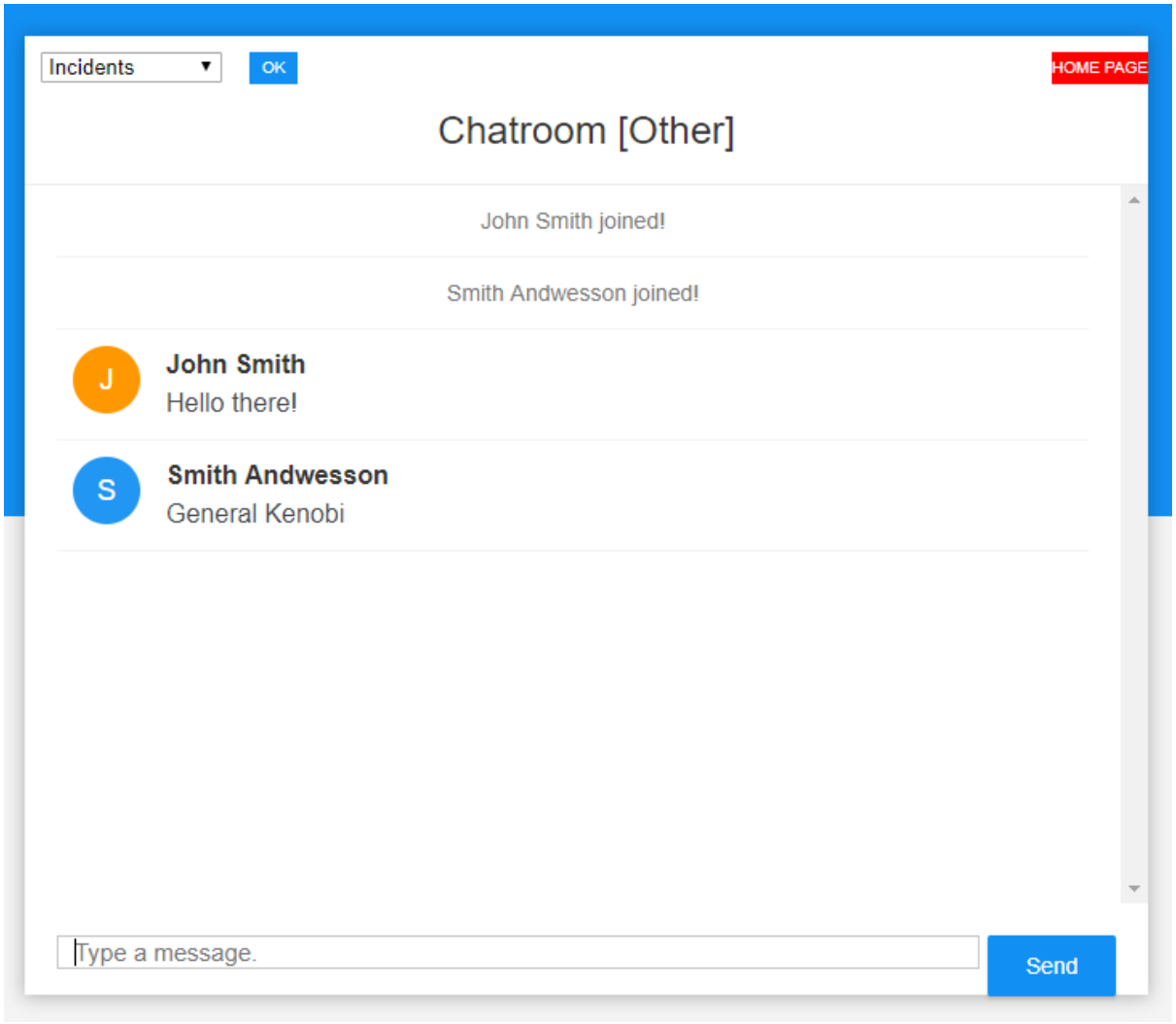
ask for

Type: Incidents

type	count
Street Light Out	4112
Tree Trim	2379
Pothole in Street	1672
Garbage Cart Black Maintenance/Replacement	1232
Abandoned Vehicle Complaint	1075
Rodent Baiting/Rat Complaint	726
Sanitation Code Violation	554
Graffiti Removal	529

Home Page

Chatting


Incidents
OK
HOME PAGE
Chatroom [Other]
John Smith joined!
Smith Andwesson joined!
J
John Smith
Hello there!
S
Smith Andwesson
General Kenobi
Type a message.
Send

Group by Zip-Code, all Incidents

Incidents per Zipcode

zip	count
UNKNOWN	1917490
60632	200937
60629	188877
60618	181708
60647	176619
60617	165293
60608	163571
60623	148840
60628	145279
60620	133585

Links από datasets

https://www.kaggle.com/chicago/chicago-food-inspections
https://www.kaggle.com/chicago/chicago-bike-racks
https://www.kaggle.com/chicago/chicago-average-daily-traffic-counts
https://www.kaggle.com/isadoraamorim/trafficcrasheschicago
https://www.kaggle.com/chicago/chicago-affordable-rental-housing-developments
https://www.kaggle.com/chicago/chicago-city-owned-land-inventory
https://www.kaggle.com/currie32/crimes-in-chicago
https://www.kaggle.com/chicago/chicago-311-service-requests

Link από κώδικα

https://drive.google.com/open?id=1FNHlappr70LZCL2h6SOPQU3ObdoNweUX

Link από demo-video:

https://drive.google.com/open?id=1zR9E3H2C0B1Sh0KIZJSQmH8P4HQc64rR