



ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

Εργασία στο μάθημα
«Υπηρεσίες και Συστήματα Διαδικτύου»
6ου εξαμήνου, 2023

ΟΑΣΑ Ticket Validations

Καζάζης Γεώργιος, it214124
Χαρίτος Δημήτριος, it21395

Ημερομηνία παράδοσης: 14/06/2023

ΠΕΡΙΕΧΟΜΕΝΑ

Εισαγωγή	3
Περιγραφή ανοικτής πηγής δεδομένων	3
Διαδικασία αποθήκευσης	4
Ανάλυση Υλοποίησης Εφαρμογής	4
Oasa-Basic-Flow Tab	4
User Management Tab	8
DB-Flow Tab	11
RabbitMQ-Flows Tab	12
Dashboard Tab	13
Οπτικοποίηση δεδομένων	14
Συμπεράσματα	18
Προβλήματα που αντιμετωπίσαμε	18

Εισαγωγή

Η παρούσα εργασία έχει υλοποιηθεί στο περιβάλλον του **Node-RED**. Αφορά την απόκτηση συλλογής ανοιχτών δεδομένων από το https://data.gov.gr/datasets/oasa_ridership/ με σκοπό την **επεξεργασία** δεδομένων που αφορούν το επιβατικό κοινό του ΟΑΣΑ, την **αποθήκευσή** τους σε μια βάση δεδομένων με τη χρήση της **sqlite**, την εφαρμογή **k-means clustering** για την **ομαδοποίηση** των δεδομένων και τελικά την **εξαγωγή συμπερασμάτων** και την **οπτικοποίηση** αυτών μέσα από ένα **dashboard**. Επίσης παρέχει τη δυνατότητα στον χρήστη να κάνει εγγραφή στο σύστημα για να μπορεί να έχει πρόσβαση στα δεδομένα του, καθώς και αποτρέπει την είσοδο σε μη εξουσιοδοτημένους χρήστες. Ο χρήστης αφού εισέλθει στο σύστημα έχει την επιλογή να εισάγει την ημερομηνία που επιθυμεί για να ενημερωθεί για το κατά πόσο υπάρχει συνωστισμός στους σημαντικότερους σταθμούς του ΟΑΣΑ για εκείνη την ημέρα και η ενημέρωση γίνεται με **γραπτό μήνυμα**, με **gauges** καθώς και ένα **barchart**. Τέλος ο χρήστης μπορεί να εισάγει άλλη ημερομηνία και να ενημερωθεί ή να **εξέλθει** από το σύστημα.

Περιγραφή ανοικτής πηγής δεδομένων

Αρχικά, η ανοιχτή πηγή δεδομένων που χρησιμοποιήθηκε προέρχεται από το site data.gov.gr στο οποίο μπορεί να γίνει εύρεση δεδομένων μέσα από 10 θεματικές ενότητες με 49 σειρές δεδομένων. Για να επιτευχθεί η πρόσβαση στο **API** πρέπει πρώτα να γίνει εγγραφή στην υπηρεσία και αίτηση για την παραλαβή ατομικού κλειδιού API. Η παραπάνω διαδικασία ολοκληρώνεται άμεσα. Το API που χρησιμοποιήθηκε ονομάζεται **Επιβατικό κοινό στον ΟΑΣΑ** και ανήκει στην κατηγορία **Μετακινήσεις**, με έναρξη καταγραφής δεδομένων στις **08/08/2020** και καθημερινή ανανέωση. Τα κυριότερα χαρακτηριστικά των δεδομένων που δίνει το API είναι οι σταθμοί του ΟΑΣΑ (μετρό, ηλεκτρικός, προαστιακός) καθώς και οι **συνολικές επικυρώσεις εισιτηρίων** που γίνονται σε κάθε σταθμό ανά ώρα, για κάθε ημέρα.

Διαδικασία αποθήκευσης

Για την αποθήκευση των δεδομένων χρησιμοποιήθηκε η **SQLite database** αφού πρώτα έγινε εγκατάσταση του πακέτου **node-red-node-sqlite** από το πεδίο **Manage palette** του Node-RED. Για την παρακολούθηση του τρόπου δημιουργίας των tables της βάσης και του τρόπου αποθήκευσης των δεδομένων έγινε χρήση του **DB Browser για SQLite** το οποίο αποτελεί ένα **open source γραφικό περιβάλλον** προβολής και επεξεργασίας αρχείων **.sqlite**.

Ανάλυση Υλοποίησης Εφαρμογής

Το συνολικό flow στο οποίο έχει στηριχθεί η εργασία συμπεριλαμβάνεται σε **πέντε tabs**.

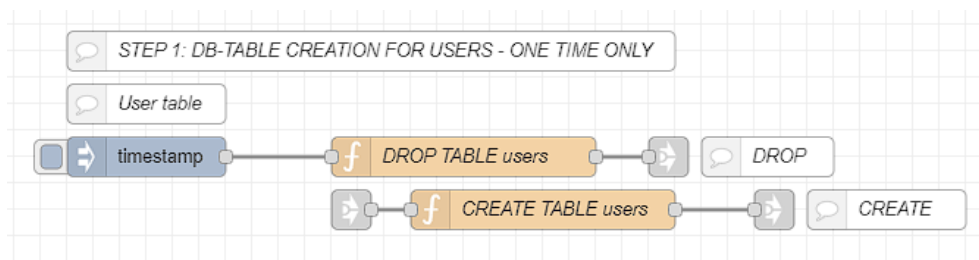
- ❖ oasa-basic-flow
- ❖ User Management
- ❖ db-flow
- ❖ rabbitmq-flows
- ❖ Dashboard

Τα nodes έχουν κατηγοριοποιηθεί κατάλληλα στα επιμέρους tabs για να είναι πιο καθαρά τα flows και περισσότερο κατανοητά. Επίσης έχουν συνδεθεί με ένα πλήθος από link-in και link-out nodes και έχουν σχολιαστεί για την επεξήγηση της λειτουργίας κάθε ροής.

Oasa-Basic-Flow Tab

☐ Στο **tab oasa-basic-flow** πραγματοποιούνται οι εξής διεργασίες:

- Διαγραφή και δημιουργία του **table users** στη βάση δεδομένων userdb.sqlite. Εκεί αποθηκεύονται όλοι οι χρήστες που κάνουν register στην εφαρμογή. (Εικόνα 1)



Εικόνα 1. Users table

- Στο παρακάτω flow υλοποιείται η εισαγωγή δεδομένων από το **Επιβατικό κοινό στον ΟΑΣΑ** API για διάστημα τριών (3) μηνών. Ο καθορισμός του διαστήματος των ημερών δίνεται από ένα **inject node** στις μεταβλητές **msg.date_from** και **msg.date_to**.

Έπειτα έχουμε τη διαγραφή και τη δημιουργία του **table clusters** στη βάση δεδομένων **stationdb.sqlite**. Εκεί αποθηκεύονται ο σταθμός, η ημέρα και οι τιμές **high**, **middle** και **low centroid** που θα υπολογιστούν μετά από το **k-means clustering** με τρία **clusters** και προκύπτουν από τα **validations** της κάθε ημέρας, σε κάθε σταθμό, για το σύνολο των ωρών.

Η διαδικασία της διαγραφής του **table clusters** και της επαναδημιουργίας του στη βάση δεδομένων γίνεται για να αποφευχθεί η ύπαρξη **διπλότυπων** εγγραφών στη βάση άρα και η αποφυγή παραγωγής λάθος συμπερασμάτων κατά την επεξεργασία των δεδομένων καθώς και στην τελική τιμή των τριών **centroids**.

Έπειτα γίνεται χρήση ενός **http request node**, με μέθοδο **GET** για να ζητήσει δεδομένα από το URL που του δίνουμε και αφορά το API. Να σημειωθεί ότι στο πεδίο **headers** δίνουμε στο **Authorization** το **API key** που έχουμε πάρει από την υπηρεσία παροχής ανοικτών δεδομένων και ότι η επιστροφή των δεδομένων γίνονται σε μορφή **parsed json object** για να χρησιμοποιηθούν μετέπειτα από τις συναρτήσεις.

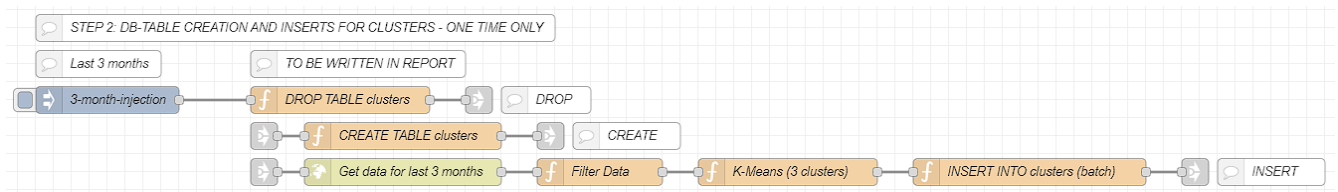
Η έξοδος του **http request node** καταλήγει στην συνάρτηση **Filter Data** στην οποία όπως λέει και το όνομά της γίνεται φιλτράρισμα των δεδομένων. Αρχικά, επιλέγουμε μια λίστα από σταθμούς τους οποίους θα εξετάσουμε και για τους οποίους θα βγάλουμε συμπεράσματα. Οι σταθμοί που επιλέχθηκαν είναι οι **"ΟΜΟΝΟΙΑ"**, **"ΣΥΝΤΑΓΜΑ"**, **"ΜΟΝΑΣΤΗΡΑΚΙ"**, **"ΣΤΑΘΜΟΣ ΛΑΡΙΣΗΣ"**, **"ΑΤΤΙΚΗ"**, **"ΑΕΡΟΔΡΟΜΙΟ"** και επιλέχθηκαν με τη λογική ότι αυτοί είναι οι πιο πολυσύχναστοι σταθμοί της Αθήνας ή αποτελούν μεγαλύτερα κέντρα διέλευσης πολιτών καθώς σε αυτούς τους σταθμούς συνδέονται παραπάνω από μια γραμμές του ΟΑΣΑ. Οπότε μελετώντας τη ροή του κόσμου σε αυτούς τους σταθμούς μπορούμε να συμπεράνουμε την γενικότερη κατάσταση σε όλο το δίκτυο του ΟΑΣΑ. Μέσα στη συνάρτηση λοιπόν ελέγχουμε το κάθε αντικείμενο που έχουμε λάβει από το API και αν αυτό ανήκει σε έναν από τους προαναφερθέντες σταθμούς, καταγράφουμε το **όνομα** του σταθμού, την **ημέρα**, την **ώρα** και τα **validations** εκείνης της ώρας και τα αποθηκεύουμε και το προωθούμε στη συνάρτηση **K-Means (3 clusters)** για να γίνει η συσταδοποίηση και η εξαγωγή των **centroids**.

Έπειτα χρησιμοποιούμε μία ακόμη συνάρτηση, την **INSERT INTO clusters** η οποία δημιουργεί το κατάλληλο **query** για την εισαγωγή δεδομένων σε **batches**, στο **table clusters**, για πιο ταχύτερη εισαγωγή, μιας και μία-μία εγγραφή έπαιρνε σημαντικά περισσότερο χρόνο. Στο **table clusters**, τα δεδομένα που θα αποθηκευτούν θα έχουν το **clusterid** (autoincrement) το **όνομα** του σταθμού, την **ημέρα** και τις **float** τιμές των **high**, **middle** και **low centroids**. (Εικόνα 2)

Οπότε για κάθε σταθμό θα υπάρχουν εφτά εγγραφές, μία για κάθε ημέρα της εβδομάδας. Και τέλος χρησιμοποιούμε ένα **sqlite node** για να αποθηκεύσουμε τα δεδομένα στην βάση μετά την επεξεργασία τους.

Έτσι ολοκληρώνεται το πρώτο flow, το οποίο εισάγει δεδομένα τριών (3) μηνών στην είσοδο και μετά από **K-means clustering** μας δείχνει ποια είναι τα τρία **centroids** για τους σταθμούς που έχουμε επιλέξει και στη συνέχεια θα χρησιμοποιήσουμε για σύγκριση με

τα validations ενός σταθμού για μία συγκεκριμένη ημέρα και την εξαγωγή συμπερασμάτων. Τέλος αποθηκεύουμε αυτά τα δεδομένα στη βάση.



Εικόνα 2. Clusters table

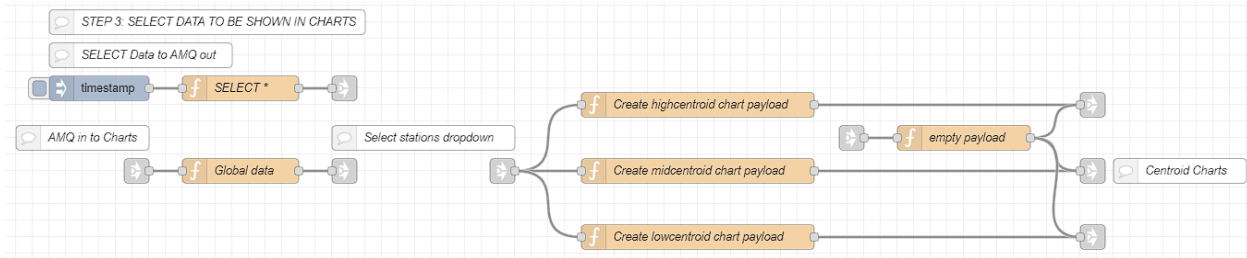
- Το επόμενο flow παίρνει όλα τα δεδομένα από το stationdb.sqlite και τα στέλνει στο **RabbitMQ** στο **oasaexchange** που έχουμε δημιουργήσει. Από εκεί τα δεδομένα επιστρέφονται και συνδέονται με τρία διαφορετικά charts, ένα για κάθε centroid.

Έτσι με τη χρήση ενός **dropdown** μπορούμε να επιλέξουμε τους σταθμούς που επιθυμούμε και να δούμε σε τι τιμές κυμαίνονται τα validations σε σχέση με τα τρία centroids, για κάθε ημέρα της εβδομάδας για την περίοδο των τριών μηνών που εξετάσαμε.

Πιο λεπτομερώς, το flow ξεκινάει με ένα injection node, το οποίο κάνει trigger την function **SELECT ***, η οποία επιλέγει όλα τα δεδομένα που έχουν αποθηκευτεί στη βάση. Στη συνέχεια ένα json node μετατρέπει τα δεδομένα σε json μορφή και αυτά θα δοθούν ως είσοδος σε ένα **amqp out** node το οποίο θα συνδεθεί με το RabbitMQ και θα στείλει εκεί τα δεδομένα της βάσης. (Εικόνα 3)

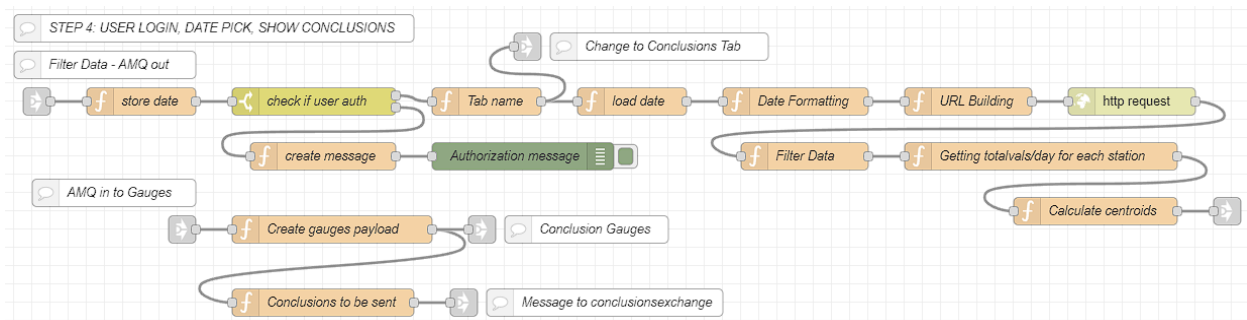
Για το amqp-out node που έχει τον ρόλο του producer έχουμε δημιουργήσει έναν broker τον **RabbitMQOASA** ο οποίος συνδέεται στο **localhost**, στο **port 5672**. Επίσης όσον αφορά τα exchange info, ο τύπος των μηνυμάτων είναι **direct** και στέλνονται στο **exchange name: oasaexchange** και στο **routing key: centroidchart** το οποίο χρησιμοποιείται από το exchange για να αποφασίσει πως θα δρομολογηθεί το μήνυμα με βάση το exchange type και τα bindings. Επίσης έχει γίνει bind του exchange με μία ουρά η οποία λειτουργεί ως buffer στα μηνύματα που λαμβάνονται στο exchange για να επιτευχθεί ασύγχρονη επικοινωνία. Στην συνέχεια χρησιμοποιούμε ένα **amqp-in** node που έχει το ρόλο του **consumer** και έχει τις ίδιες ρυθμίσεις με τον amqp-out node, ίδιο broker, direct type, exchange name και routing key για να κάνει consume τα messages που παράγονται.

Έπειτα τα δεδομένα στέλνονται στη function global data όπου γίνονται global για να χρησιμοποιηθούν αργότερα και από εκεί το flow οδηγείται στο dashboard tab και συγκεκριμένα σε ένα dropdown node **Select stations** στο οποίο επιλέγουμε τους σταθμούς που θέλουμε και από εκεί οδηγούμαστε πίσω στο **oasa-basic-flow** και στα τρία ξεχωριστά charts για το κάθε centroid. Τα οποία δεν αφήνουμε να προβληθούν από τον χρήστη αλλά μόνο από εμάς για έλεγχο επιτυχούς επεξεργασίας δεδομένων.



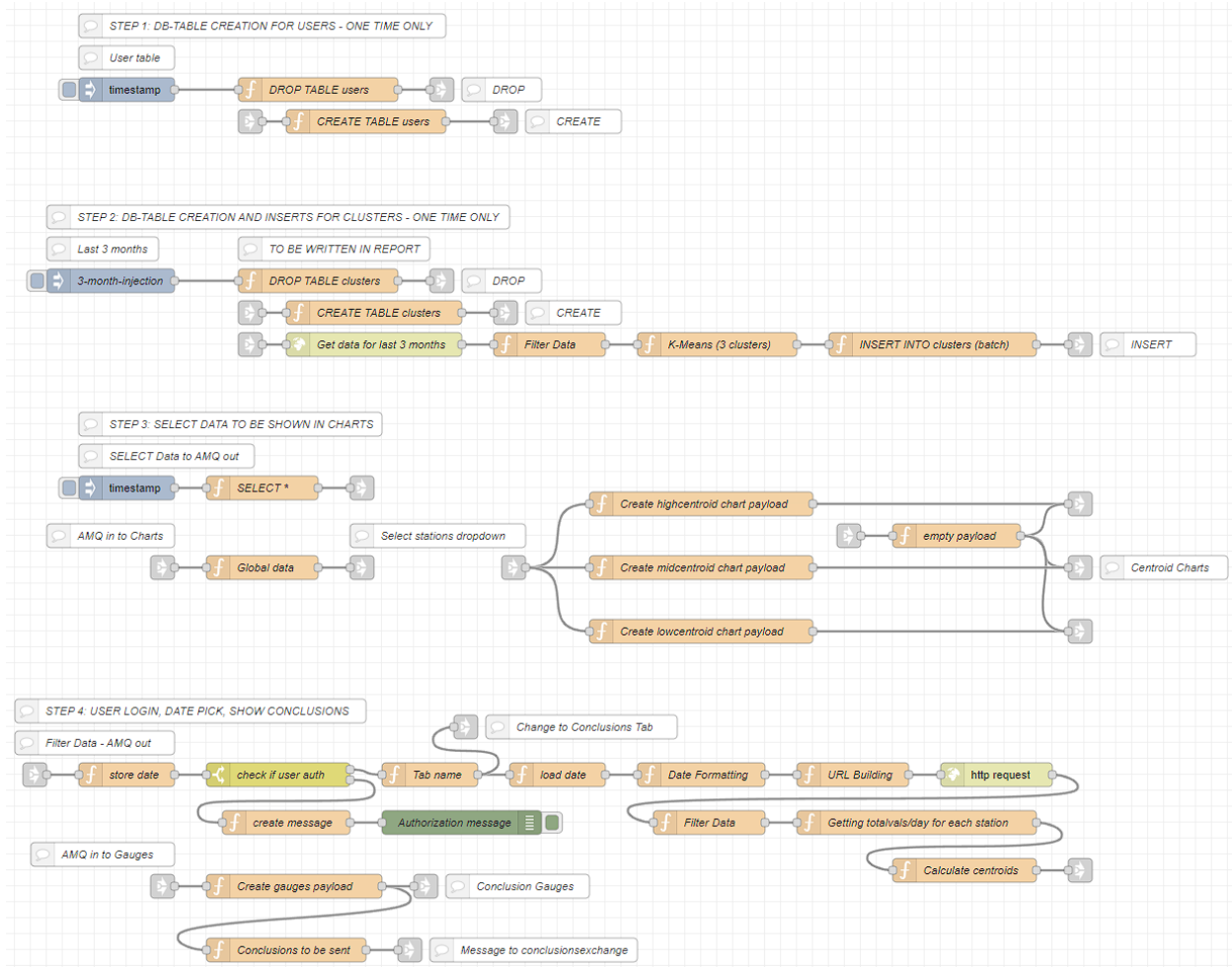
Εικόνα 3. Cluster selects

- Η τελευταία ροή στο oasa-basic-flow chart υλοποιείται αφού έχει επιτευχθεί η είσοδος του χρήστη στην εφαρμογή. Στην αρχή εισάγει την επιθυμητή ημερομηνία όπως φαίνεται στο dashboard tab μέσα από ένα date picker node και μετά με ένα link out μεταφερόμαστε στην function store date όπου κρατάμε αυτή την ημερομηνία. Στη συνέχεια ελέγχουμε αν ο χρήστης έχει τα απαραίτητα **permissions** εξετάζοντας την global μεταβλητή **auth_result** αν έχει τιμή 1 ή 0. Αν δεν έχει δεν του εμφανίζουμε απο-τελέσματα και εμφανίζουμε το αντίστοιχο μήνυμα στην καρτέλα debug. Αν έχει τα απαραίτητα δικαιώματα, ανακατευθύνεται στο tab Conclusions του dashboard. (Εικόνα 4)



Εικόνα 4. Conclusions

Συνολικά το oasa-basic-flow tab φαίνεται παρακάτω. (Εικόνα 5)



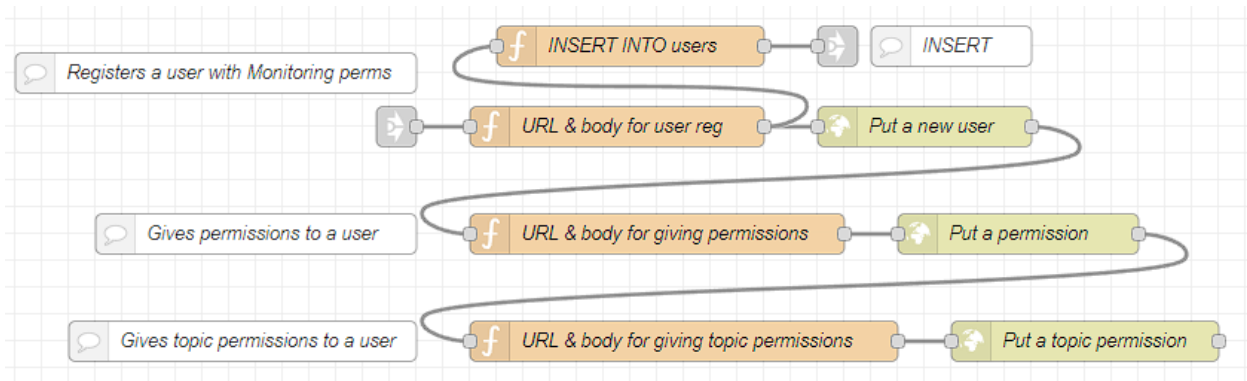
Εικόνα 5. Oasa basic flow

User Management Tab

☐ Στο **tab User Management** πραγματοποιούνται οι εξής διεργασίες:

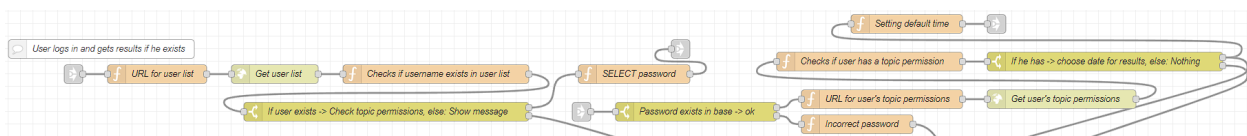
- Καθώς ένας χρήστης κάνει **register** στο σύστημα συμπληρώνοντας το **username** το οποίο πρέπει να είναι μοναδικό και το **password** του, τότε αυτός εγγράφεται ως χρήστης στο rabbitmq χρησιμοποιώντας ένα **PUT http request** στο endpoint **/api/users/name** και το **tag monitoring** στο χρήστη για να έχει δικαίωμα να κάνει monitor των δεδομένων και να μην μπορεί να τα αλλάξει. Επίσης γίνεται εγγραφή του χρήστη στη βάση **userdb**, table **users**. Στη συνέχεια, πάλι με μία function δημιουργούμε ένα url με endpoint το **/api/permissions/vhost/name**, για να δώσουμε στον χρήστη που μόλις έκανε εγγραφή **permissions** και επαναλαμβάνουμε την ίδια διαδικασία για να δώσουμε **topic permissions** στον χρήστη φτιάχνοντας url με endpoint το **/api/topic-permissions/vhost/name**.

Αν ο ίδιος έχει επιλέξει να λαμβάνει πληροφορίες για θέματα του ΟΑΣΑ, κατά την εγγραφή του (επιλέγοντας το **switch: OASA news**), τότε θα έχει δικαιώματα στο **conclusionsexchange** και θα μπορεί να δει στο dashboard τα αποτελέσματα για τη ροή στους σταθμούς του ΟΑΣΑ. Διαφορετικά, θα εγγραφεί στο **defaultexchange**, από το οποίο δεν θα μπορεί να λάβει τα αποτελέσματα. (Εικόνα 6)



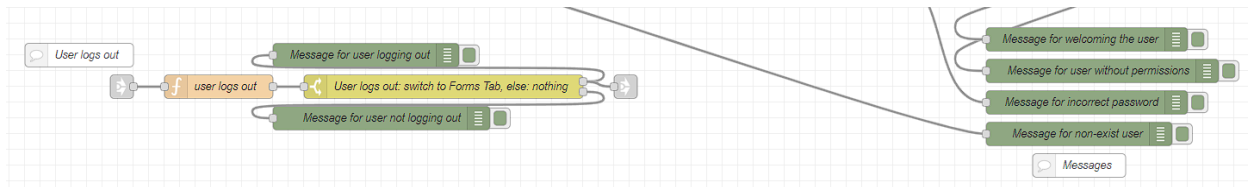
Εικόνα 6. Register user

- Κατά τη διάρκεια του login ενός χρήστη στο σύστημα γίνεται έλεγχος αν ο χρήστης είναι πράγματι εγγεγραμμένος. Αρχικά δημιουργούμε σε μία function το url με endpoint το **/api/users/** για να πάρουμε με ένα get request όλους τους χρήστες που έχουν γραφτεί στο σύστημα και βρίσκονται στο rabbitmq. Στη συνέχεια ελέγχουμε αν ο χρήστης υπάρχει. Αν όχι το **result** παραμένει μηδέν (0) και εμφανίζεται το μήνυμα ότι ο χρήστης δεν υπάρχει αλλιώς το **result** γίνεται ίσο με ένα (1) και προχωράμε από το switch node σε μία ακόμη function όπου θα γίνει ο έλεγχος του password αυτή τη φορά από τη βάση usersdb. Αν περάσει και αυτόν τον έλεγχο ο χρήστης προχωράει στον επόμενο έλεγχο για τα **permissions** που έχει αλλιώς του εμφανίζεται μήνυμα για **incorrect password**. Αν ο χρήστης δεν έχει τα κατάλληλα permissions πάλι του εμφανίζεται το αντίστοιχο μήνυμα και δεν του επιτρέπεται η οπτικοποίηση των αποτελεσμάτων, αλλιώς αν είναι γραμμένος στο **conclusionsexchange** τότε του εμφανίζεται ένα **welcoming message** και μεταφέρεται στο **conclusions tab** όπου θα επιλέξει ημερομηνία για την οποία θα δει τα αποτελέσματα. (Εικόνα 7)



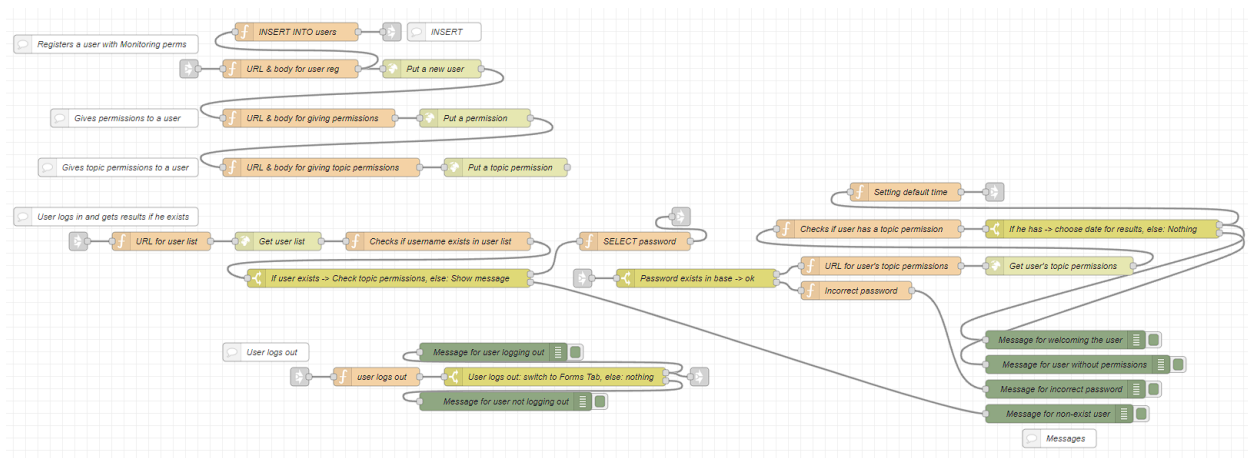
Εικόνα 7. Login user

- Επίσης υπάρχει ένα flow (Εικόνα 8) που δίνει τη δυνατότητα στον χρήστη να κάνει **log out**. Του εμφανίζει παράθυρο με τις επιλογές να παραμείνει στη σελίδα ή να ολοκληρώσει την αποσύνδεση και αντίστοιχα εμφανίζεται το κατάλληλο μήνυμα και αν αποσυνδεθεί τότε από το dashboard μεταφέρεται στην αρχική φόρμα.



Εικόνα 8. Logout user

Συνολικά το User Management tab φαίνεται παρακάτω. (Εικόνα 9)

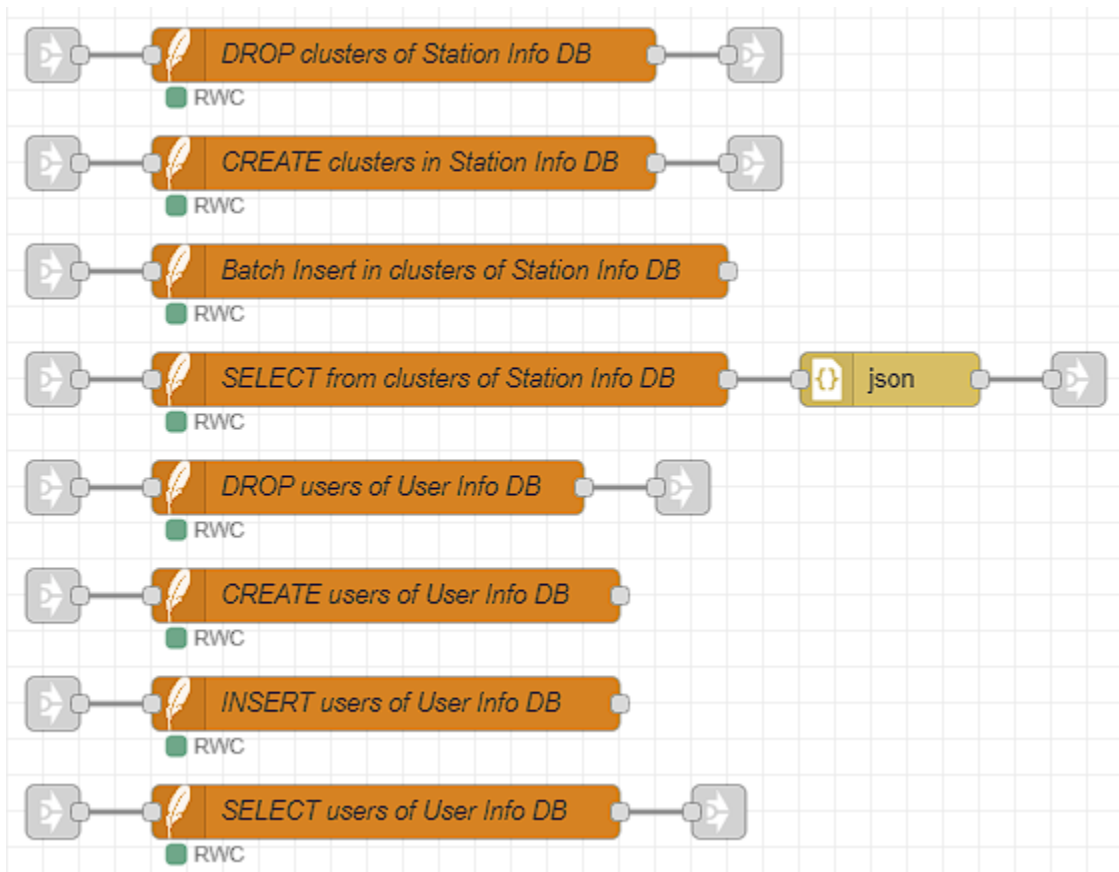


Εικόνα 9. User management flow

DB-Flow Tab

□ Στο **tab db-flow** πραγματοποιούνται οι εξής διεργασίες:

- Στο db-flow tab περιέχονται όλοι οι sqlite κόμβοι οι οποίοι συνδέονται με τις κεντρικές ροές μέσω **link-in** και **link-out** και χρησιμοποιούνται για να εκτελεστούν οι παρακάτω εντολές στις βάσεις δεδομένων: **drop**, **create**, **batch insert** για κατά πολύ ταχύτερη αποθήκευση δεδομένων, **select from** και **insert into**. (Εικόνα 10)

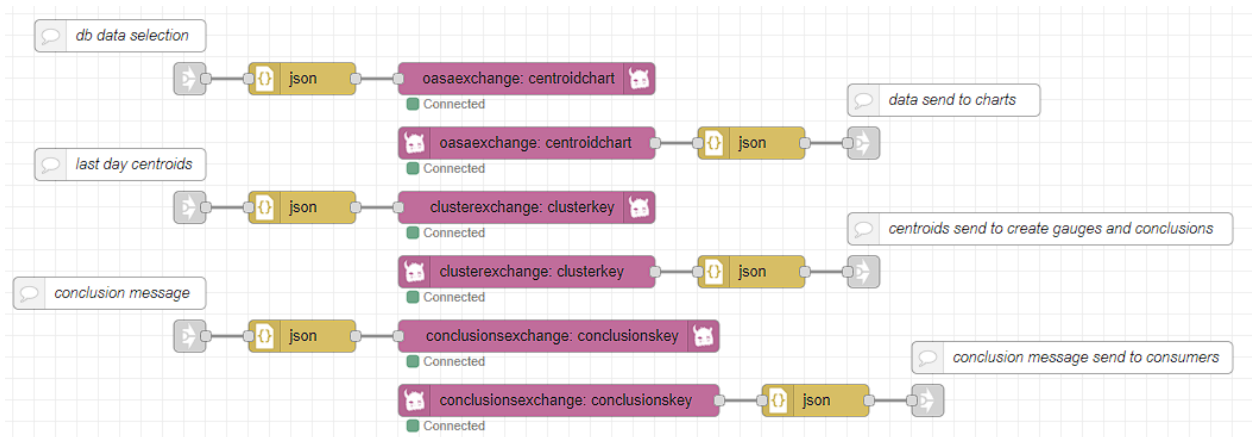


Εικόνα 10. DB flows

RabbitMQ-Flows Tab

□ Στο **tab rabbitmq-flows** πραγματοποιούνται οι εξής διεργασίες:

- Εδώ χρησιμοποιείται ένα exchange και τρία διαφορετικά routing keys για τρεις διαφορετικές διεργασίες. Το **centroidchart** συνδέεται με τα charts στο dashboard που δίνουν συνολικά τα δεδομένα των επιλεγμένων σταθμών και τις τιμές των validation για τα τρία centroids, high, middle και low που υπολογίστηκαν από το k-means clustering. (Εικόνα 11)



Εικόνα 11. RabbitMQ flows

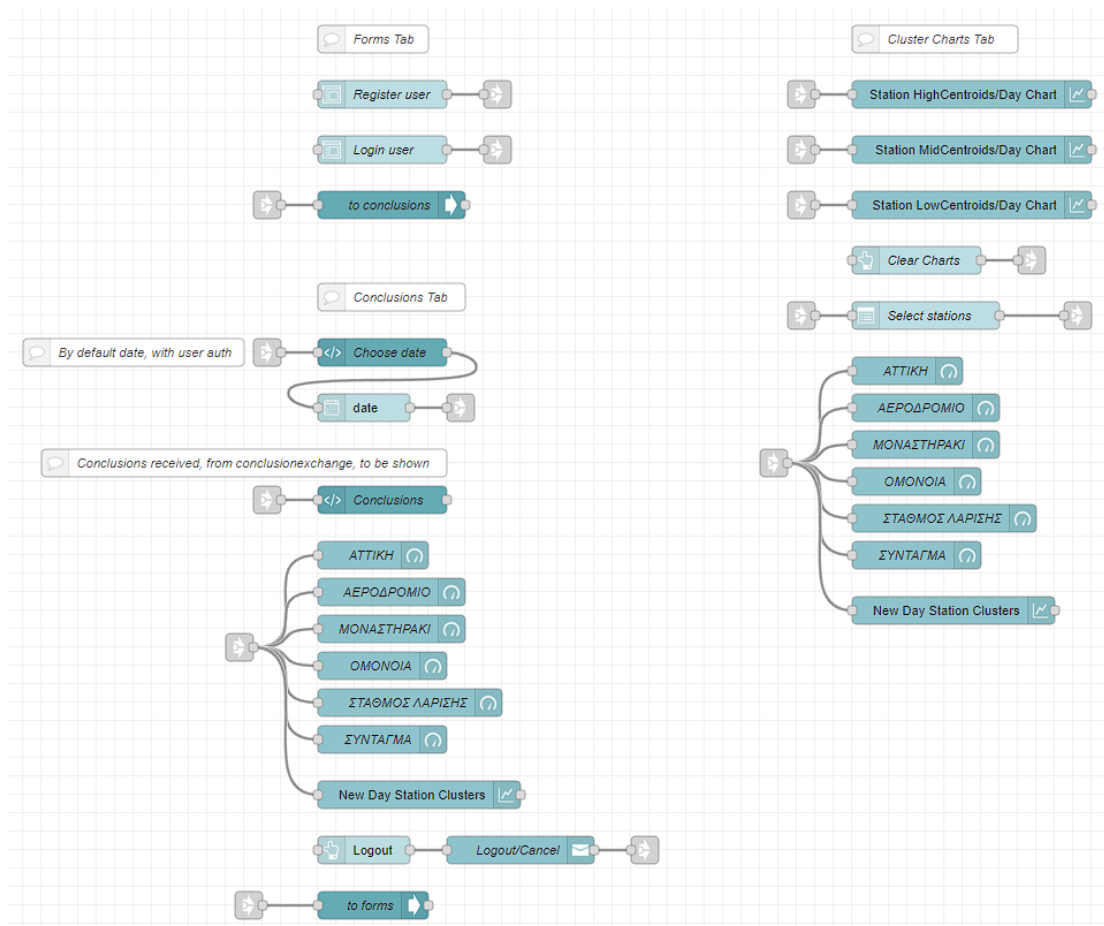
Dashboard Tab

□ Στο **tab Dashboard** πραγματοποιούνται οι εξής διεργασίες:

- Εδώ υπάρχουν όσα nodes αφορούν το dashboard. Έχουν χρησιμοποιηθεί nodes όπως **forms**, **ui control**, **date picker**, **template**, **gauge**, **button**, **notification**, **line chart**, **bar chart**.

Κάθε node συνδέεται με **link-in** και **link-out** nodes, με τις υπόλοιπες ροές, **ΒΔ**, **User Management** και το **oasa basic flow**. Τέλος, έχουν κατηγοριοποιηθεί ανάλογα με το αντίστοιχο **Tab** στο οποίο βρίσκονται. Στην επόμενη σελίδα θα δούμε και τη μορφή τους. (Εικόνες 13-)

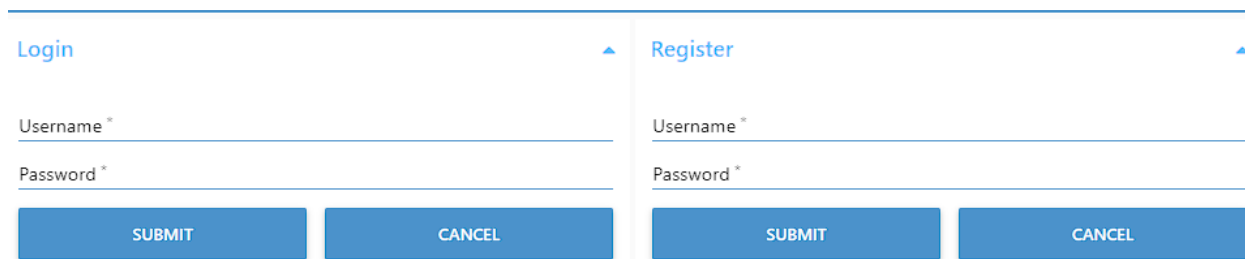
Συνολικά το Dashboard tab φαίνεται παρακάτω. (Εικόνα 12)



Εικόνα 12. Dashboard flows

Οπτικοποίηση δεδομένων

Όταν ο χρήστης χρησιμοποιεί για πρώτη φορά την εφαρμογή ξεκινάει την περιήγηση του από τις δύο φόρμες, την **register** και την **login**. Από εκεί μπορεί να κάνει εγγραφή στο σύστημα ή εφόσον έχει δημιουργήσει ήδη λογαριασμό να κάνει login με τα σωστά credentials. (Εικόνα 13)



Εικόνα 13. Register-Login forms

Μόλις ολοκληρώσει επιτυχώς το login μεταφέρεται στο **Conclusions tab** (Εικόνα 14) όπου εκεί του εμφανίζεται μήνυμα να επιλέξει την ημερομηνία για την οποία θέλει να μάθει πληροφορίες. Μόλις γίνει εισαγωγή της ημερομηνίας του εμφανίζονται τα αντίστοιχα μηνύματα που τον ενημερώνουν για την κίνηση στους επιλεγμένους σταθμούς και τα αντίστοιχα gauges τα οποία είναι **κόκκινα** (3) για μεγάλο συνωστισμό, **κίτρινα** (2) για μέση κατάσταση και **πράσινα** (1) για ελαφριά κίνηση στους σταθμούς. Επίσης στον χρήστη εμφανίζεται και ένα **bar chart** (Εικόνα 15) το οποίο του απεικονίζει περισσότερο συγκεντρωτικά τις ίδιες ενδείξεις. Ακόμη υπάρχει και το κουμπί του **Logout** στο τέλος της σελίδας, το οποίο θα τον μεταφέρει στην αρχική σελίδα με τις φόρμες.

Conclusions

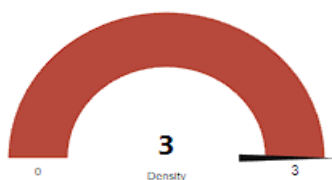
Please choose a date: (default: 01/06/2023)

date  01/06/2023

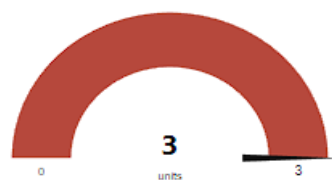
On 2023-06-01:

- Station ΑΤΤΙΚΗ was **too much crowded**
- Station ΑΕΡΟΔΡΟΜΙΟ was **too much crowded**
- Station ΜΟΝΑΣΤΗΡΑΚΙ was **a bit crowded**
- Station ΟΜΟΝΟΙΑ was **a bit crowded**
- Station ΣΤΑΘΜΟΣ ΛΑΡΙΣΗΣ was **too much crowded**
- Station ΣΥΝΤΑΓΜΑ was **too much crowded**

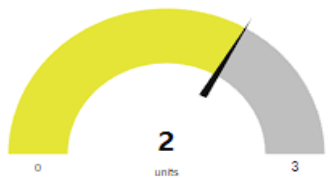
ΑΤΤΙΚΗ



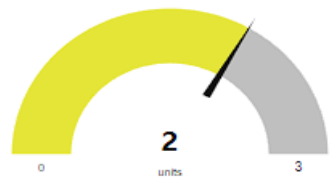
ΑΕΡΟΔΡΟΜΙΟ



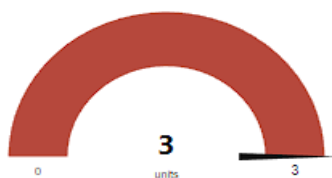
ΜΟΝΑΣΤΗΡΑΚΙ



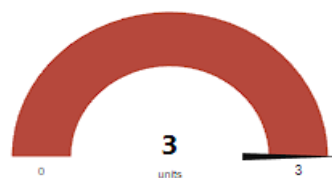
ΟΜΟΝΟΙΑ



ΣΤΑΘΜΟΣ ΛΑΡΙΣΗΣ

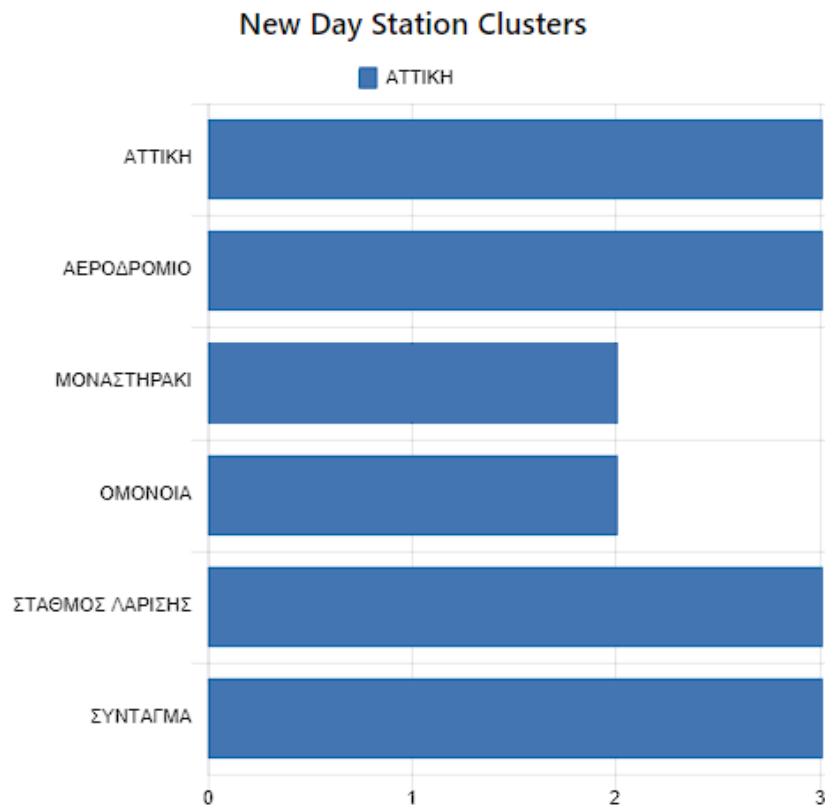


ΣΥΝΤΑΓΜΑ



LOGOUT

Εικόνα 14. Conclusions Tab - Message, Gauges, Logout

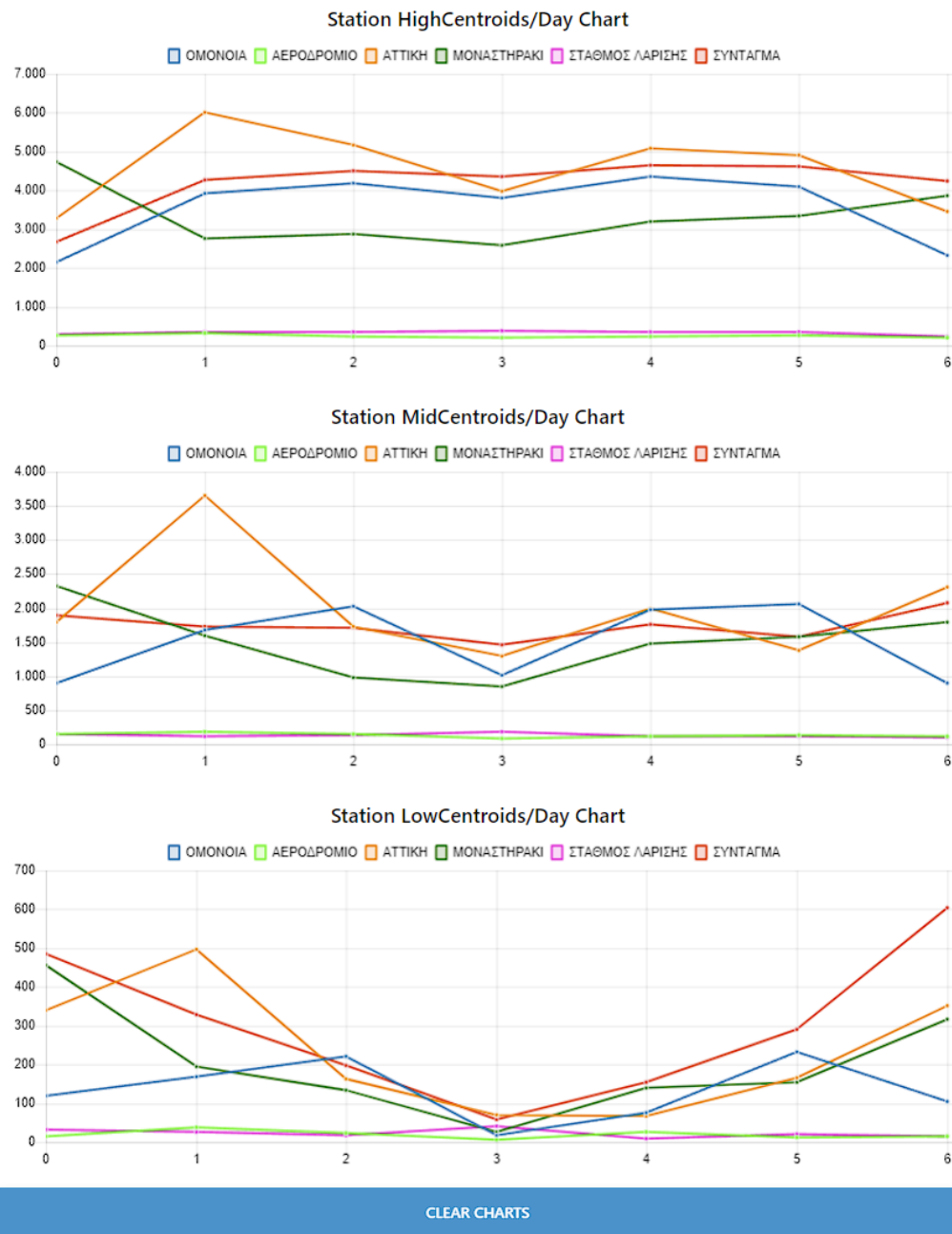


Εικόνα 15. Conclusion bar chart

Επιπρόσθετα υπάρχει η δυνατότητα προβολής των τριών **cluster charts**, ένα για το κάθε centroid που απεικονίζουν τις τιμές των validation για τους σταθμούς που θα επιλέξουμε, για την κάθε ημέρα της εβδομάδας (Κυριακή ως Σάββατο) οι οποίες έχουν αντικατασταθεί με αριθμούς με Κυριακή = 0. Την πρόσβαση σε αυτά τα charts την έχει μόνο ο **administrator** του συστήματος και χρησιμοποιούνται για τον έλεγχο της επεξεργασίας των δεδομένων και των αποτελεσμάτων του k-means clustering. (Εικόνα 16)

Station Selection

All stations shown



Εικόνα 16. Cluster charts Tab

Συμπεράσματα

Οι χρήστες που εγγράφονται σε αυτή με σκοπό να χρησιμοποιήσουν τα μέσα του ΟΑΣΑ μπορούν να εξάγουν κάποια συμπεράσματα για τις μετακινήσεις τους. Δεν υπάρχει η δυνατότητα να πραγματοποιηθεί κάποια πρόβλεψη για τις μελλοντικές ημέρες για τις οποίες δεν υπάρχουν προφανώς δεδομένα για να συλλεχθούν αλλά στην ουσία ο χρήστης μπορεί να μελετήσει τα δεδομένα που του εμφανίζονται για μια ημέρα της επιλογής του και να εξάγει ο ίδιος κάποια συμπεράσματα. Για παράδειγμα αν χρησιμοποιεί το μετρό μια συγκεκριμένη ημέρα μπορεί να παρατηρήσει σε τι τιμές συνωστισμού κυμαίνονται οι επικυρώσεις στους κεντρικούς σταθμούς τις τελευταίες τέσσερις εβδομάδες και να εξάγει ένα μοτίβο και να περιμένει ανάλογες τιμές.

Προβλήματα που αντιμετωπίσαμε

Η εργασία προχώρησε ομαλά, παρότι ήμασταν αρχάριοι στην διαχείριση του **Node-Red** και του **RabbitMQ**. Δυσκολευτήκαμε να βρούμε **API** με αποτέλεσμα να καθυστερήσουμε σε αυτό το κομμάτι, αλλά από τη στιγμή που βρήκαμε, συνεχίσαμε και ολοκληρώσαμε την εργασία εγκαίρως.

- Αυτό που πρέπει να σημειωθεί είναι ότι στο συγκεκριμένο API, δεν είχαμε **live feed** δεδομένων, μιας και κάθε μέρα ανέβαιναν τα δεδομένα της προηγούμενης ημέρας, με αποτέλεσμα το πιο πρόσφατο που θα μπορούσε να δει ο χρήστης είναι η χθεσινή μέρα.
- Επίσης υπάρχει πιθανότητα **λάθος** στις επικυρώσεις των εισιτηρίων, διότι υπάρχουν επικυρώσεις (> 0) σε ώρες που το μετρό (κυρίως στους σταθμούς που χρησιμοποιήσαμε) παραμένει κλειστό. Αυτό σίγουρα δεν οφείλεται σε κάποιο **shift** των ωρών, μιας και σε κάθε ώρα οι επικυρώσεις ήταν διάφορες του μηδενός. Όπως και να 'χει, είναι μικρό το κακό, αφού τα συμπεράσματα τα βγάζουμε από τις μέρες συνολικά, και όχι από τις ώρες.
- Επιλέξαμε να ασχοληθούμε μόνο με τους 6 σημαντικότερους (κατά τη γνώμη μας) και πιο πολυσύχναστους σταθμούς, το οποίο έχει ως αποτέλεσμα να μην είναι τόσο δυναμική η εφαρμογή, και να αφορά μόνο χρήστες που θα χρησιμοποιήσουν μετρό σε αυτούς τους συγκεκριμένους σταθμούς.

Ευχαριστούμε!

[Github repo](#)