



Πτυχιακή Εργασία

Δημιουργία λογισμικού παρακολούθησης, απεικόνισης και εμφάνισης
ειδοποιήσεων δεξαμενών δικτύου ύδρευσης

Ιωάννης Μυρώνης

Π2018221

Επιβλέπων : Μάρκος Αυλωνίτης

Κέρκυρα Ιανουάριος 2023

Επιβλέπων

Μάρκος Αυλωνίτης, Αναπληρωτής Καθηγητής,
Ιόνιο Πανεπιστήμιο

Τριμελής Επιτροπή

Μάρκος Αυλωνίτης, Αναπληρωτής Καθηγητής, Ιόνιο Πανεπιστήμιο

Ιωάννης Καρύδης, Επίκουρος Καθηγητής, Ιόνιο Πανεπιστήμιο

Αλέξανδρος Πανάρετος, ΕΔΙΠ, Ιόνιο Πανεπιστήμιο

Περίληψη

Η πτυχιακή εργασία με τίτλο «Δημιουργία λογισμικού παρακολούθησης, απεικόνισης και εμφάνισης ειδοποιήσεων για δεξαμενές δικτύου ύδρευσης » αφορά την υλοποίηση μιας web εφαρμογής μέσω της οποίας θα γίνει οπτικοποίηση των δεξαμενών ενός δικτύου ύδρευσης ώστε να είναι δυνατή η παρακολούθηση και επομένως η διαχείρισή του. Πιο συγκεκριμένα, η υλοποίηση αυτής της εφαρμογής, έχει σαν σκοπό την απεικόνιση της κατάστασης κάθε δεξαμενής του δικτύου ύδρευσης σε πραγματικό χρόνο.

Μέσω της εφαρμογής, οι χρήστες του δικτύου που είναι οι υπάλληλοι της εταιρίας ύδρευσης, θα μπορούν να ενημερώνονται για την ποσότητα του νερού που υπάρχει σε κάθε δεξαμενή του συστήματος καθώς και για την ποσότητα του νερού που εισρέει στην δεξαμενή σε πραγματικό χρόνο. Εκτός από την οπτικοποίηση των δεξαμενών του συστήματος, η εφαρμογή προσφέρει και άλλες χρήσιμες λειτουργικότητες. Οι χρήστες ενημερώνονται μέσω ηχητικών και οπτικών ειδοποιήσεων με μηνύματα για τυχόν προβλήματα που προκύπτουν και αφορούν σε υπερχειλίση ή σε έλλειψη νερού σε κάποια από τις δεξαμενές του συστήματος. Επιπρόσθετα, η εφαρμογή προσφέρει στατιστικά στοιχεία για την κατανάλωση της κάθε δεξαμενής του δικτύου προσφέροντας σημαντικές πληροφορίες στους διαχειριστές του συστήματος. Επιπλέον, δίνει την δυνατότητα στον χρήστη να καθορίσει το ελάχιστο και μέγιστο επίπεδο ενεργοποίησης του συναγερμού. Τέλος, η εφαρμογή είναι διαθέσιμη σε δύο γλώσσες, Αγγλικά και Ελληνικά, και μέσω του μενού ο χρήστης επιλέγει όποια από τις δύο επιθυμεί.

Για την δημιουργία της εφαρμογής, αξιοποιήθηκαν δεδομένα που λαμβάνονται από δύο διαφορετικά είδη αισθητήρων: τους αισθητήρες που είναι τοποθετημένοι μέσα στις δεξαμενές του δικτύου ύδρευσης, οι οποίοι μετράνε τον όγκο του περιεχόμενου νερού, και τους αισθητήρες που είναι τοποθετημένοι στην είσοδο κάθε δεξαμενής και μετράνε τον όγκο του εισερχόμενου νερού. Στη συνέχεια, όλα τα δεδομένα των αισθητήρων αποθηκεύονται σε μια βάση δεδομένων, στον αντίστοιχο πίνακα που έχει δημιουργηθεί για κάθε μία από τις δεξαμενές του συστήματος. Τέλος, υπάρχει και ένας πίνακας που είναι προϊόν επεξεργασίας των δεδομένων και αφορά τα στατιστικά στοιχεία.

Η υλοποίηση της διαδικτυακής αυτής εφαρμογής, έγινε με χρήση Html, Css και JavaScript όσον αφορά το κομμάτι του χρήστη (frontend) και τη λειτουργικότητα αυτού. Η βάση δεδομένων για την αποθήκευση των πληροφοριών από τις δεξαμενές έχει δημιουργηθεί με PostgreSQL ενώ το backend τμήμα που αφορά τη διασύνδεση του χρήστη με την βάση δεδομένων υλοποιήθηκε με χρήση εξυπηρετητή NodeJS.

Λέξεις κλειδιά: Δίκτυα ύδρευσης, Γραφική διεπαφή, Διαδικτυακή εφαρμογή, Πελάτης εξυπηρετητής

Abstract

The thesis entitled "Creation of software for monitoring, visualization and displaying alerts for water supply network tanks" refers to the implementation of a web application whereby the visualization of the water supply network tanks will be accomplished in order to have better control of the water stored in them. This application aims to visualize the status of each tank of the water supply network in real time.

Through the application, the employees of the water supply (who are the administrators of the system) will be able to get informed on the amount of water present in each tank of the system as well as the amount of water flowing into the tank in real time. Apart from the visualization of the water tank, the application offers other useful functionalities. Users are informed through audio and visual alarm notifications regarding any problems that arise concerning overflow or lack of water in any of the system's tanks. Moreover, the application provides statistics records for the consumption of each and every tank. In addition it allows the user to set the minimum and maximum level of alarms. Finally, the application is available in two languages and using a menu the user can choose the language they require.

In order to create this application, data obtained from sensors installed inside the water supply network tanks, is used to measure the volume of water. Furthermore, data obtained from sensors placed at the inlet of each tank, is used to measure the volume of water flowing into the tanks. The data from the sensors is stored in a database. Data of each tank of the system, are stored in a separate table in the database. There is also a table that is the result of data processing and is related to statistics.

The web application was implemented using HTML, CSS and JavaScript, in terms of the user part (frontend) and its functionality. The database was created using PostgreSQL and the backend part concerning the user interface with the database, was implemented using a NodeJS server.

Πρόλογος και Ευχαριστίες

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω θερμά για την πολύτιμη βοήθεια που μου προσέφεραν κατά τη διάρκεια πραγματοποίησης αυτής της εργασίας, τον υπεύθυνο καθηγητή κύριο Μ. Αυλωνίτη και την ομάδα του.

Κέρκυρα, Δεκέμβριος 2022

Ιωάννης Μυρώνης

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ.....	8
Κεφάλαιο 1	10
Εισαγωγή	10
Κεφάλαιο 2	12
Δίκτυο ύδρευσης και αισθητήρες.....	12
2.1 Γενικά περί Δικτύων ύδρευσης.....	12
2.2 Αρχή λειτουργίας αισθητήρων	13
2.2.1 Αισθητήρες υπερήχων	13
2.2.2 Αισθητήρες ροής νερού	14
Κεφάλαιο 3	16
Λογισμικό της Εφαρμογής.....	16
3.1 Βάση Δεδομένων PostgreSQL.....	16
3.1.1 Εργαλείο Διαχείρισης pgAdmin.....	17
3.2 Εξυπηρετητής NodeJS	18
3.2.1 Γενικά	18
3.2.2 Βιβλιοθήκες του NodeJS	18
3.3 Τμήμα εφαρμογής που αλληλεπιδρά με τον χρήστη (Client side).....	19
Κεφάλαιο 4	22
Υλοποίηση της Εφαρμογής.....	22
4.1 Γενική Περιγραφή της Εφαρμογής	22
4.2 Διάγραμμα Ροής Δεδομένων	24
4.3 Βάση Δεδομένων PostgreSQL.....	25
4.3.1 Πίνακας δεξαμενής Γαστούρι	26
4.3.2 Πίνακας δεξαμενής Κάβος.....	26
4.3.2 Πίνακας καταναλώσεων	27
4.4 Εξυπηρετητής NodeJS	28
4.4.1 Δημιουργία του Εξυπηρετητή	28
4.4.2 RoutePath/ currentPreviousMeasurementsTank1.....	28
4.4.3 RoutePath/currentPreviousMeasurementsTank2	30
4.4.4 RoutePath/ consumptionRegistration	30

4.4.5	RoutePath / statisticsHourlyTank1	31
4.4.6	RoutePath / statisticsDailyTank1	32
4.4.7	RoutePath / statisticsHourlyTank2	33
4.4.8	RoutePath / statisticsDailyTank2	34
4.5	Ανάπτυξη του τμήματος της εφαρμογής που αλληλεπιδρά με τον χρήστη (Front-End)	35
4.5.1	Μενού περιήγησης	35
4.5.2	Χάρτης.....	36
4.5.3	Εμφάνιση δεξαμενών.....	37
4.5.4	Γραμμή κατάστασης.....	39
4.5.5	Καθορισμός επιπέδων συναγερμού	40
4.5.6	Στατιστικά.....	40
4.5.7	Επιλογή γλώσσας	42
4.5.8	About	42
4.6	Δεδομένα	43
4.7	Εγκατάσταση λογισμικού εφαρμογής.....	44
4.7.1	Βάση Δεδομένων PostgreSQL	44
4.7.2	Εξυπηρετητής NodeJS.....	44
4.8	Προβλήματα κατά την υλοποίηση της εφαρμογής	45
	Κεφάλαιο 5	48
	Συμπεράσματα- Μελλοντικές Τροποποιήσεις.....	48
	Βιβλιογραφία	50
	Συντμήσεις.....	51
	Γλωσσάρι Ξενικών Όρων	52
	ΠΑΡΑΡΤΗΜΑ.....	53

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

Εικόνα 1: Αισθητήρας υπερήχων	13
Εικόνα 2: Αισθητήρας ροής νερού	14
Εικόνα 3: Στιγμιότυπο της βάσης μέσω του εργαλείου διαχείρισης pgAdmin.....	17
Εικόνα 4: Διάγραμμα ροής δεδομένων	25
Εικόνα 5: Μενού περιήγησης.....	36
Εικόνα 6: Βασικός χάρτης εφαρμογής	37
Εικόνα 7: Γραφική αναπαράσταση δεξαμενής	38
Εικόνα 8: Γραφική αναπαράσταση δεξαμενής σε κατάσταση alarm.....	39
Εικόνα 9: Φόρμα εισαγωγής επιπέδων συναγερμού	40
Εικόνα 10: κουμπιά επιλογής στατιστικών	41
Εικόνα 11: Ραβδογράμματα μέσης ημερήσιας και εβδομαδιαίας κατανάλωσης.....	42
Εικόνα 12: Φόρμα about.....	43

Κεφάλαιο 1

Εισαγωγή

Το θέμα της πτυχιακής εργασίας αφορά την δημιουργία ενός λογισμικού μέσω του οποίου θα είναι δυνατή η παρακολούθηση ενός δικτύου ύδρευσης σε πραγματικό χρόνο και η ενημέρωση του διαχειριστή του δικτύου σε περίπτωση δυσλειτουργίας. Έχοντας εντοπίσει τις ανάγκες των διαχειριστών του δικτύου θα υλοποιήσουμε με βάση αυτές την εφαρμογή μας.

Πιο συγκεκριμένα οι δεξαμενές του δικτύου ύδρευσης, είναι διάσπαρτες σε διάφορα σημεία του νησιού της Κέρκυρας. Κάθε δεξαμενή τροφοδοτείται με νερό μέσω αντλιών που είναι εγκατεστημένες στην είσοδο κάθε δεξαμενής. Το πρόβλημα λοιπόν που έχει προκύψει, είναι η αδυναμία παρακολούθησης της λειτουργίας της αντλίας ώστε να είναι εφικτός ο έγκαιρος εντοπισμός ενδεχόμενης βλάβης της, καθώς και η παρακολούθηση της ποσότητας νερού που υπάρχει σε κάθε δεξαμενή σε πραγματικό χρόνο. Η αδυναμία γνώσης της κατανάλωσης νερού κάθε δεξαμενής καθιστά αδύνατο τον προγραμματισμό λειτουργίας των αντλιών με βάση τις καταναλώσεις, κάτι που θα βοηθούσε στην αδιάλειπτη παροχή νερού και συνεπώς στη βελτίωση της ποιότητας του παρεχόμενου προϊόντος.

Η λύση στο πρόβλημα, δόθηκε με την εγκατάσταση δύο αισθητήρων. Οι αισθητήρες τοποθετήθηκαν από το τμήμα πληροφορικής του Ιόνιου Πανεπιστήμιου και πιο συγκεκριμένα από το εργαστήριο CMODLab, στο πλαίσιο ενός προγράμματος. Ο ένας αισθητήρας τοποθετήθηκε στην είσοδο της δεξαμενής με σκοπό τη μέτρηση της ποσότητας του νερού που εισρέει στη δεξαμενή, ο δεύτερος αισθητήρας τοποθετήθηκε

Κεφάλαιο 1 : Εισαγωγή

μέσα στη δεξαμενή ώστε να μετράει την ποσότητα του νερού που περιέχεται σε αυτή. Έχοντας λάβει τα δεδομένα των αισθητήρων κάθε δεξαμενής, τα οποία είναι αποθηκευμένα σε μία βάση δεδομένων του εργαστηρίου CMODLab, και κάνοντας την κατάλληλη επεξεργασία, εισάγονται στη βάση δεδομένων που έχει δημιουργηθεί για την υλοποίηση της εφαρμογής.

Ακολουθώντας, μέσω της εφαρμογής, πραγματοποιείται η οπτική απεικόνιση κάθε δεξαμενής ώστε να εμφανίζει την ποσότητα νερού που περιέχει σε πραγματικό χρόνο. Επιπρόσθετα, παρέχεται οπτικό και ηχητικό alarm προς ενημέρωση του ο χρήστη, σε περίπτωση που η στάθμη του νερού είναι πολύ υψηλή ή πολύ χαμηλή. Ο χρήστης, μέσω του μενού της εφαρμογής, έχει τη δυνατότητα να τροποποιήσει τα επίπεδα ενεργοποίησης του alarm. Τέλος, προσφέρονται στατιστικά στοιχεία σχετικά με την κατανάλωση που υπάρχει σε κάθε δεξαμενή. Μέσω των στατιστικών, οι χειριστές του συστήματος, θα μπορούν να έχουν εικόνα των καταναλώσεων και άρα καλύτερη διαχείριση του δικτύου προσαρμοσμένη στις πραγματικές ανάγκες των χρηστών. Η εμφάνιση και δημιουργία των στατιστικών, γίνεται με ραβδογράμματα τα οποία εμφανίζονται μετά από επιλογή από τον χρήστη της ημερομηνίας που επιθυμεί. Η εφαρμογή προσφέρει στατιστικά που αφορούν τη μέση τιμή της κατανάλωσης ανά ημέρα ή εβδομάδα. Τέλος, η εφαρμογή δίνει στο χειριστή, τη δυνατότητα να επιλέξει γλώσσα από το μενού της εφαρμογής. Οι δύο προσφερόμενες γλώσσες είναι Αγγλικά και Ελληνικά.

Κεφάλαιο 2

Δίκτυο ύδρευσης και αισθητήρες

Για να μπορέσει να γίνει κατανοητή η ανάγκη υλοποίησης της εφαρμογής Water Tank ώστε να είναι δυνατή η παρακολούθηση της σωστής λειτουργίας των δεξαμενών ενός δικτύου ύδρευσης, κρίνεται σκόπιμο να γίνει μια σύντομη αναφορά στον τρόπο λειτουργίας και στα τμήματα από τα οποία αποτελείται ένα δίκτυο. Επιπλέον, θα γίνει μια αναφορά στην αρχή λειτουργίας των αισθητήρων που χρησιμοποιήθηκαν για τη συλλογή των δεδομένων, βάσει των οποίων υλοποιήσαμε την εφαρμογή.

2.1 Γενικά περί Δικτύων ύδρευσης

Ένα δίκτυο ύδρευσης, εκτός των αγωγών το μήκος των οποίων καλύπτει εκατοντάδες ή ακόμη και χιλιάδες χιλιόμετρα, αποτελείται από δεξαμενές και αντλιοστάσια. Συγκεκριμένα, για την υδροδότηση των περιοχών που βρίσκονται σε μεγάλο υψόμετρο, η χρήση των αντλιοστασίων είναι απαραίτητη ώστε να μπορεί το νερό να φτάσει σε δεξαμενές που βρίσκονται διάσπαρτες συνήθως σε μεγάλα υψόμετρα. Στις δεξαμενές αποθηκεύονται χιλιάδες κυβικά μετρά νερού και η απόσταση μεταξύ τους είναι αρκετές εκατοντάδες μέτρα.

Γίνεται εύκολα αντιληπτό ότι ένα σύστημα ύδρευσης, αποτελεί ένα τεράστιο σε όγκο δίκτυο, η παρακολούθηση λειτουργίας του οποίου δεν είναι εύκολη. Λαμβάνοντας υπόψη ότι η κατασκευή του μπορεί να έχει αρχίσει δεκαετίες πριν και ότι γίνονται συνεχώς προσθήκες νέων τμημάτων, η παρακολούθησή του από ένα σύγχρονο σύστημα ελέγχου, γίνεται ακόμη πιο δύσκολη.

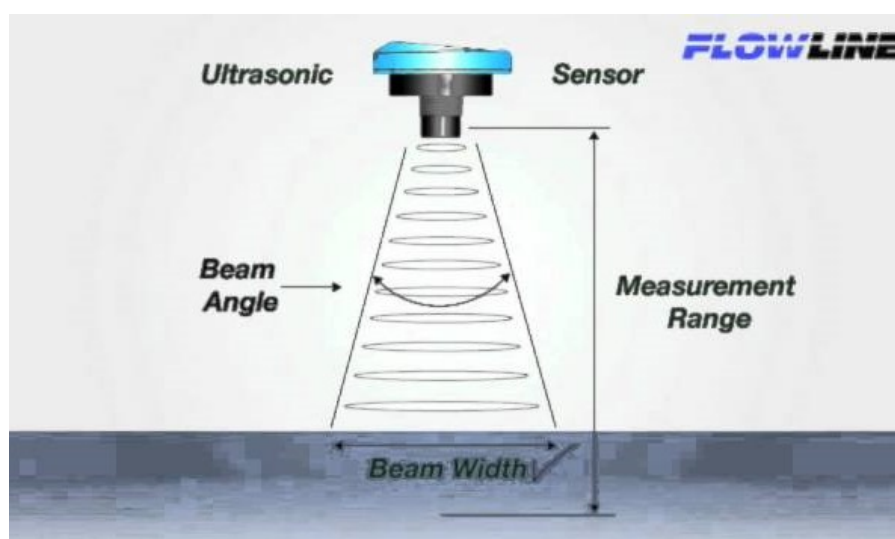
2.2 Αρχή λειτουργίας αισθητήρων

Σε αυτό το σημείο, γίνεται σύντομη περιγραφή της αρχής λειτουργίας των αισθητήρων, τα δεδομένα των οποίων χρησιμοποιούνται για την υλοποίηση της εργασίας. Για τη μέτρηση του όγκου του νερού που βρίσκεται στις δεξαμενές, χρησιμοποιήθηκαν αισθητήρες υπερήχων (ultrasonic sensors) και για τη μέτρηση του νερού που εισέρχεται στις δεξαμενές, αισθητήρες ροής νερού (water flow sensors).

2.2.1 Αισθητήρες υπερήχων

Ο αισθητήρας υπερήχων είναι ένα όργανο με τη βοήθεια του οποίου μετράμε την απόσταση από ένα αντικείμενο, με χρήση υπερηχητικών κυμάτων. Ένας αισθητήρας υπερήχων, χρησιμοποιεί έναν μετατροπέα τόσο για την αποστολή όσο και για τη λήψη υπερηχητικών παλμών που αναμεταδίδουν πληροφορίες σχετικά με την εγγύτητα ενός αντικειμένου. Τα ηχητικά κύματα υψηλής συχνότητας, ανακλώνται από την επιφάνεια (στόχος) και παράγουν διακριτά μοτίβα.

Οι αισθητήρες υπερήχων, λειτουργούν στέλνοντας ένα ηχητικό κύμα με χρήση συχνοτήτων οι οποίες βρίσκονται πάνω από το φάσμα της ανθρώπινης ακοής. Σε κάποιους αισθητήρες υπερήχων, υπάρχει μόνο ένας μετατροπέας που λειτουργεί ως μικρόφωνο για να λαμβάνει και να στέλνει τον υπερηχητικό ήχο. Σε άλλου τύπου αισθητήρες, η αποστολή γίνεται από διαφορετικό τμήμα του αισθητήρα σε σχέση με τη λήψη. Τέλος, ο αισθητήρας προσδιορίζει την απόσταση από έναν στόχο, μετρώντας τις χρονικές περιόδους μεταξύ της αποστολής και της λήψης του υπερηχητικού παλμού.

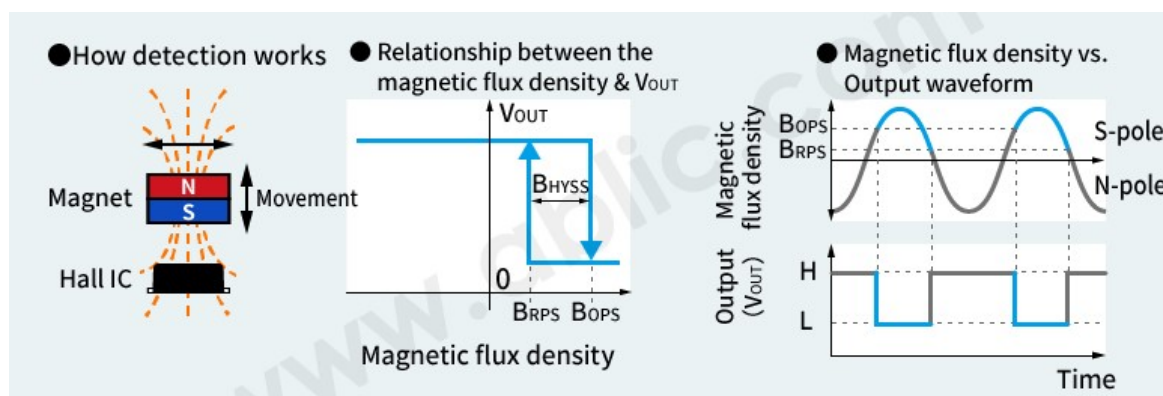


Εικόνα 1: Αισθητήρας υπερήχων

Στη συνέχεια, γνωρίζοντας την απόσταση του αισθητήρα από την επιφάνεια του νερού, εφαρμόζεται κατάλληλος αλγόριθμος υπολογισμού του όγκου του νερού που βρίσκεται μέσα στην δεξαμενή. Ο αλγόριθμος υπολογισμού, εξαρτάται από το σχήμα και το μέγεθος της κάθε δεξαμενής.

2.2.2 Αισθητήρες ροής νερού

Ο αισθητήρας ροής νερού, χρησιμοποιείται για τη μέτρηση του ρυθμού ροής του νερού. Ο ρυθμός ροής νερού, είναι ο όγκος του υγρού που εισέρχεται στη δεξαμενή ανά μονάδα χρόνου (λίτρα/λεπτό). Οι αισθητήρες ροής, βασίζονται σε μια ποικιλία διαφορετικών αρχών λειτουργίας. Ένας από τους πιο συνηθισμένους τύπους αισθητήρων ροής, είναι οι αισθητήρες Hall. Ένας αισθητήρας Hall, αποτελείται από έναν τροχό τουρμπίνας και ένα μαγνήτη. Η εισαγωγή του νερού, γίνεται από την είσοδο του αισθητήρα και η εξαγωγή από την έξοδο. Η ροή του νερού, δίνει ώθηση στον τροχό ώστε να αρχίσει να περιστρέφεται και μαζί με τον τροχό, αρχίζει να περιστρέφεται και ο μαγνήτης. Η περιστροφή του μαγνητικού πεδίου, ενεργοποιεί τον αισθητήρα Hall. Η έξοδος του αισθητήρα, αποτελείται από τετραγωνικά κύματα (τετραγωνικούς παλμούς) υψηλού και χαμηλού επιπέδου.



Εικόνα 2: Αισθητήρας ροής νερού

Κεφάλαιο 3

Λογισμικό της Εφαρμογής

Σε αυτό το κεφάλαιο παρουσιάζεται το λογισμικό που έχει επιλεγεί για την υλοποίηση της εφαρμογής Water Tanks.

3.1 Βάση Δεδομένων PostgreSQL

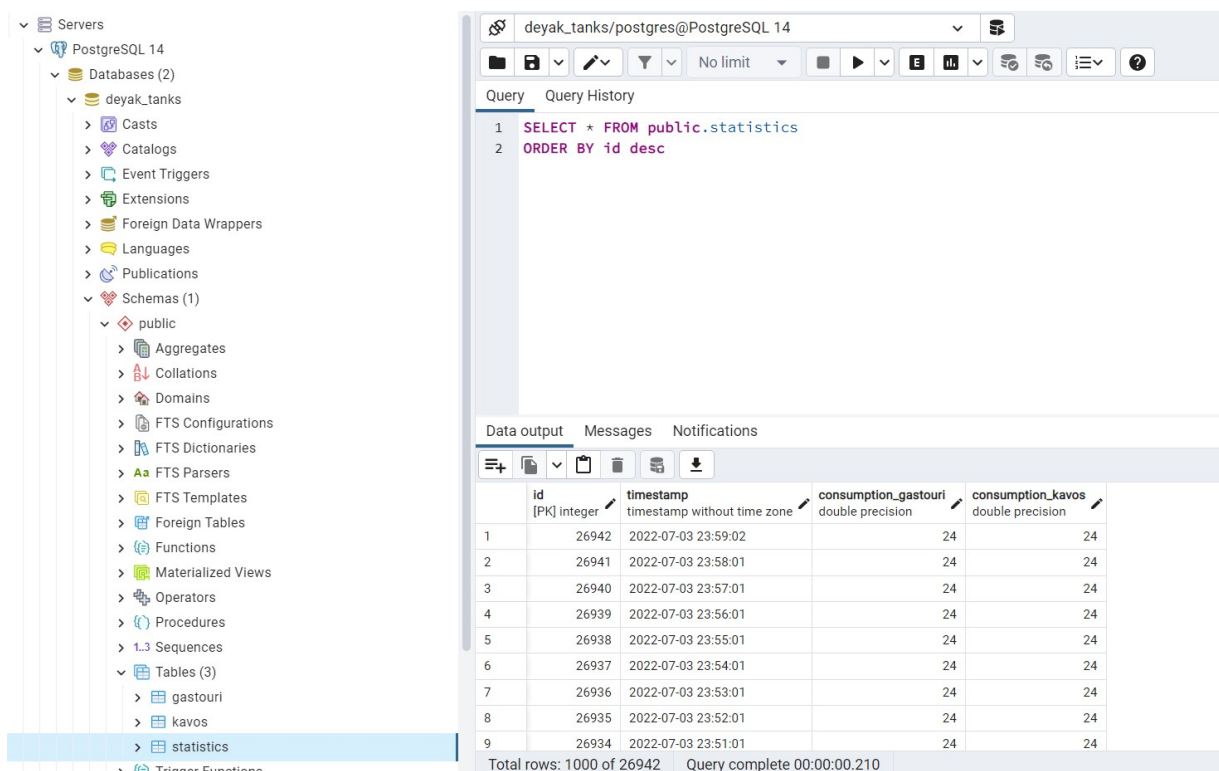
Η PostgreSQL είναι μια προηγμένη σχεσιακή βάση δεδομένων ανοικτού κώδικα που υποστηρίζει SQL αλλά και JSON ερωτήματα. Αποτελεί ένα εξαιρετικά σταθερό σύστημα διαχείρισης βάσεων δεδομένων, του οποίου η ανάπτυξη βασίζεται στην κοινότητα. Λόγω αυτού του χαρακτηριστικού, η PostgreSQL διαθέτει υψηλά επίπεδα ανθεκτικότητας, ακεραιότητας και ορθότητας. Η PostgreSQL χρησιμοποιείται ως χώρος αποθήκευσης δεδομένων για πολλές εφαρμογές διαδικτύου. Το Postgres.app περιλαμβάνει το psql, έναν πελάτη γραμμής εντολών για την PostgreSQL. Έκτος από τη γραμμή εντολών, υπάρχουν και γραφικοί πελάτες που είναι διαθέσιμοι για την PostgreSQL. Το πιο γνωστό γραφικό εργαλείο, είναι το pgAdmin. Το pgAdmin αποτελεί έναν client ανοικτού κώδικα που προσφέρει στον χρήστη τεράστιες δυνατότητες για τη διαχείριση της βάσης. Ένα ακόμη βασικό χαρακτηριστικό της PostgreSQL είναι η

Κεφάλαιο 3 : Λογισμικό εφαρμογής

συμβατότητά της με όλα τα βασικά λειτουργικά συστήματα. Τέλος, θα πρέπει να αναφέρουμε ότι η εργασία υλοποιήθηκε με χρήση της έκδοσης 14.

3.1.1 Εργαλείο Διαχείρισης pgAdmin

Το pgAdmin αποτελεί ένα ευρέως χρησιμοποιούμενο εργαλείο διαχείρισης βάσεων δεδομένων για την PostgreSQL το οποίο απλοποιεί τη διαδικασία δημιουργίας, συντήρησης και χρήσης των αντικειμένων της βάσης δεδομένων. Το εργαλείο pgAdmin προσφέρει στον χρήστη ένα πλούσιο σύνολο χαρακτηριστικών για τη διαχείριση βάσεων δεδομένων, συμπεριλαμβανομένου ενός οδηγού σύνδεσης, ενός επεξεργαστή μέσω του οποίου μπορεί να γίνει εισαγωγή ερωτημάτων SQL και ενός μηχανισμού αυτόματης δημιουργίας ερωτημάτων SQL. Το γραφικό περιβάλλον του χρήστη, έχει σχεδιαστεί με τέτοιο τρόπο ώστε να είναι φιλικό προς αυτόν. Τέλος, το pgAdmin μπορεί να χρησιμοποιηθεί είτε μέσω της διεπαφής web ή ως εφαρμογή που μπορεί να εγκατασταθεί τοπικά.



The screenshot displays the pgAdmin 4 web interface. On the left, the 'Servers' tree shows a PostgreSQL 14 server with a database named 'deyak_tanks'. The 'public' schema is selected, and the 'statistics' table is highlighted. The main pane shows the SQL query: `SELECT * FROM public.statistics ORDER BY id desc`. Below the query, the 'Data output' tab is active, showing a table with 10 rows and 4 columns: `id` (integer), `timestamp` (timestamp without time zone), `consumption_gastouri` (double precision), and `consumption_kavos` (double precision). The status bar at the bottom indicates 'Total rows: 1000 of 26942' and 'Query complete 00:00:00.210'.

	id	timestamp	consumption_gastouri	consumption_kavos
1	26942	2022-07-03 23:59:02	24	24
2	26941	2022-07-03 23:58:01	24	24
3	26940	2022-07-03 23:57:01	24	24
4	26939	2022-07-03 23:56:01	24	24
5	26938	2022-07-03 23:55:01	24	24
6	26937	2022-07-03 23:54:01	24	24
7	26936	2022-07-03 23:53:01	24	24
8	26935	2022-07-03 23:52:01	24	24
9	26934	2022-07-03 23:51:01	24	24

Εικόνα 3: Στιγμιότυπο της βάσης μέσω του εργαλείου διαχείρισης pgAdmin

3.2 Εξυπηρετητής NodeJS

3.2.1 Γενικά

Η διασύνδεση του client side τμήματος της εφαρμογής μας με τη βάση δεδομένων στην οποία είναι αποθηκευμένα τα δεδομένα που λαμβάνονται από τους αισθητήρες, γίνεται μέσω του εξυπηρετητή NodeJS, ο οποίος παρέχει περιβάλλον εκτέλεσης σε JavaScript. Αποτελεί λογισμικό ανοικτού κώδικα και είναι cross-platform, δηλαδή μπορεί να εκτελεστεί σε διαφορετικές υπολογιστικές πλατφόρμες. Αναπτύχθηκε το 2009 και βασίζεται στην μηχανή V8 JavaScript η οποία αποτελεί τον πυρήνα του φυλλομετρητή Chrome της Google. Το τεράστιο πλεονέκτημα που προσφέρει ο NodeJS σε εκατομμύρια προγραμματιστές frontend, είναι ότι πλέον μπορούν να γράφουν σε JavaScript τον κώδικα και στην πλευρά του πελάτη χωρίς να χρειάζεται να μάθουν μια εντελώς διαφορετική γλώσσα.

Επιπλέον, η ασύγχρονη και καθοδηγούμενη από συμβάντα φύση του NodeJS, τον καθιστά κατάλληλο για τη δημιουργία εφαρμογών που μπορούν να χειριστούν πολλές συνδέσεις ταυτόχρονα, κάτι που αποτελεί μια συνήθη εργασία για διακομιστές ιστού.

3.2.2 Βιβλιοθήκες του NodeJS

Ένας από τους βασικούς παράγοντες που οδήγησε σε επιτυχία το NodeJS, είναι το npm. Το npm είναι ο διαχειριστής πακέτων Node μέσω του οποίου οι προγραμματιστές JavaScript μπορούν να μοιράζονται χρήσιμα πακέτα με γρήγορο και εύκολο τρόπο.

Το npm (Node Package Manager) αποτελείται από δύο βασικά μέρη: α) ένα εργαλείο CLI (μία διεπαφή γραμμής εντολών) για τη δημοσίευση και τη λήψη των βιβλιοθηκών (πακέτων) β) ένα διαδικτυακό αποθετήριο που φιλοξενεί πακέτα JavaScript.

Βιβλιοθήκες οι οποίες έχουν αναπτυχθεί από την κοινότητα των προγραμματιστών για τον NodeJS, φιλοξενούνται στο αποθετήριο <https://www.npmjs.com/>. Η εφαρμογή που έχει δημιουργηθεί στο πλαίσιο της εργασίας, χρησιμοποιεί τα ακόλουθα πακέτα NodeJS

- **express**

Το Express είναι το πιο δημοφιλές πλαίσιο ιστού του NodeJS. Παρέχει μηχανισμούς για την αντιμετώπιση σχεδόν κάθε προβλήματος ανάπτυξης ιστού. Χτισμένο πάνω στην ενσωματωμένη

μονάδα http του NodeJS, το Express μας βοηθά να ρυθμίσουμε τη δρομολόγηση και να χειριστούμε τον κύκλο αίτησης/απόκρισης.

- **Path**

Το NodeJS παρέχει το πρόσθετο path μέσω του οποίου γίνεται πιο εύκολη η αλληλεπίδραση με τις διαδρομές αρχείων. Ακόμη, το path διαθέτει πολλές χρήσιμες ιδιότητες και μεθόδους για την πρόσβαση και τον χειρισμό των διαδρομών στο σύστημα αρχείων.

- **body-parser**

Ο Express body-parser είναι μια βιβλιοθήκη npm που χρησιμοποιείται για την επεξεργασία των δεδομένων που αποστέλλονται μέσω του σώματος (body) ενός αιτήματος http. Πιο συγκεκριμένα, ο parser, μέσω των middlewares του express, αναλύει το σώμα ενός http αιτήματος σε test, JSON, url-encoded και ακατέργαστα δεδομένα, για να ελέγξει αν πληροί κάποια κριτήρια ώστε να επιστρέφει στον ελεγκτή-στόχο,

- **fs**

Το module fs μας παρέχει μια σειρά από λειτουργίες για την αλληλεπίδραση με το σύστημα των αρχείων. Μέσω συναρτήσεων όπως writeFile(), readFile(), μας επιτρέπει να γράφουμε και να διαβάζουμε αρχεία.

- **pg**

Το node-postgres, ή pg είναι μια συλλογή από modules του NodeJS για τη διασύνδεση με μια βάση δεδομένων PostgreSQL. Μεταξύ των πολλών χαρακτηριστικών που υποστηρίζει το node-postgres, περιλαμβάνονται callbacks, promises, async/await, connection pooling, prepared statements, cursors, rich type parsing και C/C++ bindings.

- **Cors**

Το cors είναι μια βιβλιοθήκη του NodeJS που μας επιτρέπει να χαλαρώσουμε ένα χαρακτηριστικό ασφαλείας των προγράμματος περιήγησης που περιορίζει τα cross-origin αιτήματα HTTP.

3.3 Τμήμα εφαρμογής που αλληλεπιδρά με τον χρήστη (Client side)

Παράλληλα με την επιλογή των δομικών στοιχείων του backend τμήματος της εφαρμογής, δηλαδή της βάσης δεδομένων και του εξυπηρετητή, έγινε και ο σχεδιασμός του client side τμήματος της εφαρμογής. Η επιλογή του τρόπου με τον οποίο κάναμε την οπτικοποίηση των δεδομένων, είχε ως βασικό κριτήριο την επιλογή μας να σχεδιάσουμε μια web εφαρμογή. Οι σχεδιαστικές επιλογές που προσφέρονται για τη

Κεφάλαιο 3 : Λογισμικό εφαρμογής

δημιουργία μιας web εφαρμογής, είναι πάρα πολλές. Ο σχεδιαστής μπορεί να επιλέξει ανάμεσα από μια πληθώρα βιβλιοθηκών ή frameworks της Javascript ώστε να εκμεταλλευτεί κατά το μέγιστο δυνατό, τις πολλές δυνατότητες της γλώσσας. Για τον σχεδιασμό των στατιστικών που προσφέρει η εφαρμογή Water Tanks, έγινε χρήση της βιβλιοθήκης D3.

Κεφάλαιο 4

Υλοποίηση της Εφαρμογής

Σε αυτό το κεφάλαιο περιγράφεται ο τρόπος υλοποίησης της εφαρμογής Water Tanks με τη βοήθεια διαγραμμάτων ροής. Επιπλέον, θα δούμε τα επιμέρους κομμάτια της εφαρμογής πως υλοποιήθηκαν και πως επικοινωνούν μεταξύ τους. Τέλος, θα παρουσιάσουμε πληροφορίες σχετικά με την εγκατάσταση του λογισμικού που χρησιμοποιήθηκε.

4.1 Γενική Περιγραφή της Εφαρμογής

Η εφαρμογή αρχίζει με την εκτέλεση του αρχείου main.js. Η πρώτη συνάρτηση που καλείται είναι η Calculation Registration of Consumption. Μέσω της Calculation Registration of Consumption, γίνεται κλήση της Get Timestamp, η οποία επιστρέφει το timestamp της χρονικής στιγμής κατά την οποία κλήθηκε. Στη συνέχεια, μέσω της Calculation Registration of consumption, εκτελείται η συνάρτηση Get Current And Previous Values με όρισμα το αρχικό timestamp (initial timestamp). Στο επόμενο βήμα, η συνάρτηση Get Current And Previous Values ζητάει μέσω http request από τον NodeJS να φέρει από τη βάση δεδομένων τις δύο τελευταίες τιμές του water volume και την τελευταία του water rate από τον πίνακα Gastouri με κριτήριο το initial timestamp.

Έχοντας την τελευταία τιμή του water volume που έχει καταχωρηθεί στη βάση, μπορούμε να υπολογίσουμε το ποσοστό πλήρωσης της δεξαμενής πολλαπλασιάζοντας την τιμή του water volume με το εκατό και διαιρώντας το με τη μέγιστη χωρητικότητα της δεξαμενής όπως φαίνεται στον παρακάτω τύπο.

$$\text{Filling Level} = \frac{\text{water volume} \times 100}{\text{Max volume of Tank}}$$

Κεφάλαιο 4 : Υλοποίηση της εφαρμογής

Ακολούθως, εκτελείται η διαδικασία που μόλις περιγράψαμε και για την δεξαμενή Κάβος. Στη συνέχεια, γίνεται έλεγχος μέσω της Check Alarms αν οι νέες τιμές που λάβαμε για τη στάθμη του νερού των δεξαμενών, παραβιάζουν τα επίπεδα ενεργοποίησης του συναγερμού. Αν αυτή η συνθήκη ικανοποιείται, γίνεται ενεργοποίηση του συναγερμού.

Στο επόμενο βήμα, οι τιμές που έχουμε λάβει για κάθε δεξαμενή από τη βάση, περνούν ως όρισμα στη συνάρτηση Calculation Consumption όπου γίνεται ο υπολογισμός της κατανάλωσης ανά λεπτό. Ο υπολογισμός της κατανάλωσης γίνεται μετατρέποντας την τιμή του water rate από λίτρα σε κυβικά εκατοστά και προσθέτοντας την τιμή αυτή με την προτελευταία τιμή του water volume. Αφαιρούμε από αυτό το άθροισμα, την τιμή του τελευταίου water volume και η τιμή που προκύπτει μας δίνει την κατανάλωση σε κυβικά ανά λεπτό.

$$\text{consumption per minute} = (\text{previous water volume} + \text{water rate}) - \text{last water volume}$$

Ακολούθως, οι τιμές της κατανάλωσης για κάθε δεξαμενή μαζί με το timestamp που λάβαμε από το αρχικό request (Get Current And Previous Values) και το οποίο έχουμε αποθηκεύσει σε μια μεταβλητή, περνούν ως ορίσματα στην Consumption Registration και από εκεί, μέσω http request, στον NodeJS. Ο server αποστέλλει ένα query ώστε να γίνει εισαγωγή των δεδομένων στον πίνακα statistics της βάσης.

Μετά την ολοκλήρωση της Get Current And Previous Values η κλήση της οποίας έγινε με όρισμα το initial timestamp, καλείται η Set Interval. Set Interval αποτελεί μια πολύ βασική συνάρτηση για την εφαρμογή μας και εκτελείται συνεχώς κάθε εξήντα δευτερόλεπτα. Η Set Interval αρχικά λαμβάνει ένα timestamp καλώντας την get timestamp. Με όρισμα αυτό το time stamp, καλεί την Get Current And Previous Values η οποία επαναλαμβάνει ολόκληρη τη διαδικασία που περιγράψαμε πιο πάνω. Μέσω της επαναλαμβανόμενης κλήσης της Set Interval, η εφαρμογή μας παρέχει σε πραγματικό χρόνο, πληροφορίες σχετικά με τον τρέχοντα όγκο του νερού μέσα στην δεξαμενή, την κατανάλωση ανά λεπτό, την τρέχουσα ποσότητα νερού που μπαίνει στη δεξαμενή και τέλος, το ποσοστό πλήρωσης

Μια επιπλέον βασική εφαρμογή είναι η Click Tank η οποία εκτελείται κάθε εξήντα δευτερόλεπτα καθώς η κλήση της γίνεται μέσα από την Check Levels που αναφέραμε προηγουμένως. Η Click Tank καλείται επίσης και μέσω της επιλογής μιας δεξαμενής από τον χρήστη. Σκοπός της είναι να λαμβάνει και να ανανεώνει τις τιμές κάθε δεξαμενής. Έτσι, σε περίπτωση επιλογής από τον χρήστη μιας δεξαμενής, η Click Tank καλεί την Draw Tank η οποία πραγματοποιεί την γραφική απεικόνιση της δεξαμενής που έχει επιλεγεί, απεικονίζοντας την πραγματική ποσότητα του νερού που περιέχει εκείνη την στιγμή η δεξαμενή. Η Draw Tank καλείται με όρισμα την τιμή της ποσότητας που έχει η δεξαμενή μέσα.

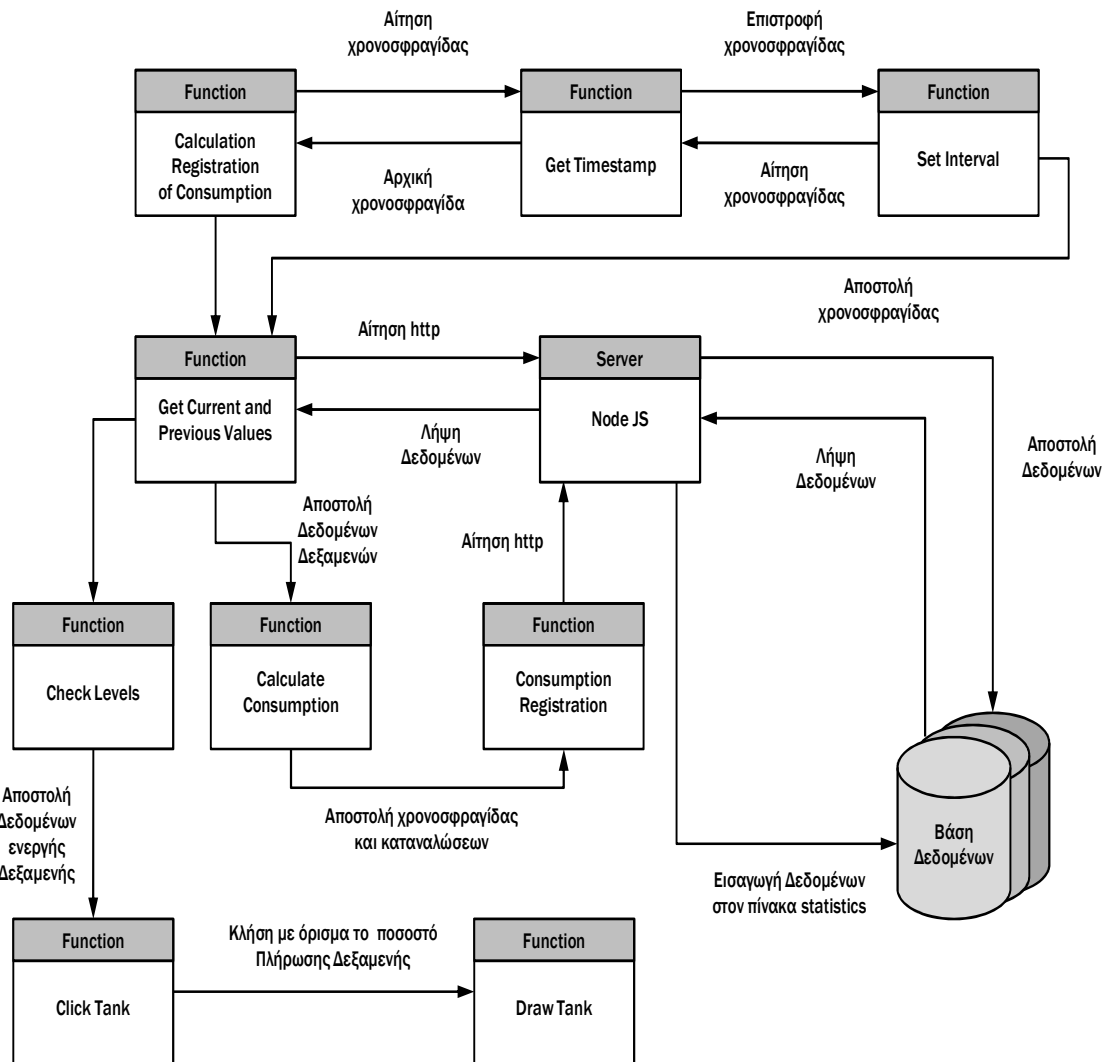
Κεφάλαιο 4 : Υλοποίηση της εφαρμογής

Τέλος, η εφαρμογή Calculate Hourly Consumption χρησιμοποιείται ώστε να αποστέλλονται στον server η αρχική και η τελική χρονοσφραγίδα για τον υπολογισμό της μέσης ημερήσιας και εβδομαδιαίας κατανάλωσης. Στο αρχείο d3form.js υπάρχουν δύο Calculate Hourly Consumption. Η διαφοροποίηση μεταξύ τους έχει να κάνει με τον τρόπο που υπολογίζουν την αρχή και το τέλος του χρονικού διαστήματος που θέλουμε ο NodeJS να αναζητήσει στην βάση.

4.2 Διάγραμμα Ροής Δεδομένων

Στο Διάγραμμα Ροής Δεδομένων παρουσιάζονται οι κυριότερες διεργασίες της εφαρμογής καθώς και τα δεδομένα που ανταλλάσσονται μεταξύ τους.

Κεφάλαιο 4 : Υλοποίηση της εφαρμογής



Εικόνα 4: Διάγραμμα ροής δεδομένων

4.3 Βάση Δεδομένων PostgreSQL

Αρχικά, υλοποιήθηκε η βάση δεδομένων με την PostgreSQL και στη συνέχεια, δημιουργήθηκε ένας πίνακας για κάθε μια από τις δεξαμενές που αποτελούν μέρος του δικτύου ύδρευσης. Επιπλέον φτιάχτηκε ένας επιπλέον πίνακας μέσα στον οποίο θα αποθηκεύονται τα στατιστικά που έχουν να κάνουν με τις καταναλώσεις ανά λεπτό για κάθε μια από τις δεξαμενές.

4.3.1 Πίνακας δεξαμενής Γαστούρι

Ο πρώτος πίνακας που δημιουργήθηκε είναι ο πίνακας Gastouri. Αρχικά έγινε η λήψη των δεδομένων από τους αισθητήρες που έχουν τοποθετηθεί στη δεξαμενή σε μορφή csv. Στη συνέχεια, για να είναι δυνατή η εισαγωγή τους στον πίνακα, έγινε η κατάλληλη επεξεργασία. Η επεξεργασία έγινε με χρήση του editor sublime text. Ο πίνακας Gastouri αποτελείται από τέσσερα πεδία. Το ID το οποίο αποτελεί και primary key του πίνακα, το timestamp τύπου timestamp, το Water_Volume τύπου double precision και το Water_Rate τύπου real. Στο πεδίο Water_Volume αποθηκεύονται οι τιμές που αφορούν την ποσότητα νερού που υπάρχει σε κάθε χρονοθυρίδα στη δεξαμενή, μετρούμενη σε κυβικά μέτρα. Στο πεδίο Water_Rate αποθηκεύονται οι τιμές του νερού που εισέρχεται στη δεξαμενή σε κάθε χρονοθυρίδα, μετρούμενη σε λίτρα. Ακολουθεί ένα ενδεικτικό παράδειγμα σύνταξης σε SQL, της εντολής με την οποία δημιουργούμε τον πίνακα Gastouri και εισάγουμε τα δεδομένα των αισθητήρων σε αυτόν.

```
CREATE TABLE GASTOURI
```

```
(ID SERIAL PRIMARY KEY,
```

```
TIMESTAMP TIMESTAMP,
```

```
WATER_VOLUME DOUBLE PRECISION,
```

```
WATER_RATE REAL);
```

```
INSERT INTO GASTOURI (TIMESTAMP, WATER_VOLUME, WATER_RATE)
```

```
VALUES ('2022-07-25 19:31:02',127.985095263,3400.0),
```

4.3.2 Πίνακας δεξαμενής Κάβος

Ο δεύτερος πίνακας που δημιουργήθηκε είναι ο πίνακας Kavos. Το αρχείο δεδομένων από το εργαστήριο, περιείχε δεδομένα μόνο από μια δεξαμενή, αυτήν του Γαστουρίου. Για να είναι δυνατός ο έλεγχος της σωστής λειτουργίας της εφαρμογής, έπρεπε να δημιουργηθεί τουλάχιστον μια ακόμη δεξαμενή. Έτσι, τα δεδομένα της δεξαμενής Kavos, αποτελούν προϊόν επεξεργασίας που βασίστηκε στα δεδομένα της δεξαμενής Gastouri. Η επεξεργασία και στα δεδομένα αυτού του πίνακα, έγινε με χρήση του editor

Κεφάλαιο 4 : Υλοποίηση της εφαρμογής

sublime text. Ο πίνακας Kavos αποτελείται και αυτός από τέσσερα πεδία ίδια με αυτά του πίνακα Gastouri. Το ID το οποίο αποτελεί και primary key του πίνακα, το timestamp τύπου timestamp, το Water_Volume τύπου double precision και το Water_Rate τύπου real. Ακολουθεί η εντολή δημιουργίας του πίνακα Kavos σε SQL.

```
CREATE TABLE KAVOS
```

```
(ID SERIAL PRIMARY KEY,
```

```
TIMESTAMP TIMESTAMP,
```

```
WATER_VOLUME DOUBLE PRECISION,
```

```
WATER_RATE REAL);
```

Στη συνέχεια, βλέπουμε παράδειγμα της εντολής sql με την οποία εισάγουμε τα δεδομένα των αισθητήρων στον πίνακα Kavos.

```
INSERT INTO KAVOS (TIMESTAMP, WATER_VOLUME, WATER_RATE)
```

```
VALUES
```

```
('2022-07-25 19:31:02',57.985095263,3400.0)
```

4.3.2 Πίνακας καταναλώσεων

Ο τρίτος και τελευταίος πίνακας που δημιουργήθηκε στη βάση δεδομένων, είναι ο πίνακας Statistics. Στον πίνακα Statistics καταχωρούνται τα δεδομένα που προκύπτουν από την κατανάλωση νερού την οποία έχουμε υπολογίσει μέσω κατάλληλου αλγόριθμου. Ο πίνακας statistics αποτελείται από τέσσερα πεδία. Το ID το οποίο αποτελεί και primary key του πίνακα, το timestamp τα δεδομένα του οποίου είναι τύπου timestamp, το CONSUMPTION_GASTOURI τύπου double precision και το CONSUMPTION_KAVOS τύπου double precision. Ακολουθεί η εντολή δημιουργίας του πίνακα σε SQL.

```
CREATE TABLE STATISTICS
```

```
(ID SERIAL PRIMARY KEY,
```

TIMESTAMP TIMESTAMP,

CONSUMPTION_GASTOURI DOUBLE PRECISION,

CONSUMPTION_KAVOS DOUBLE PRECISION);

Τέλος, βλέπουμε ένα παράδειγμα της εντολής sql με την οποία εισάγουμε τα δεδομένα των αισθητήρων στον πίνακα statistics.

INSERT INTO STATISTICS (TIMESTAMP, CONSUMPTION_GASTOURI, CONSUMPTION_KAVOS)

VALUES ('2022-07-25 19:31:02', 9.550221999999998, 2.5960099999999926);

4.4 Εξυπηρετητής NodeJS

4.4.1 Δημιουργία του Εξυπηρετητή

Σε πρώτη φάση, έγινε εγκατάσταση της βιβλιοθήκης express και ακολούθως, δημιουργήθηκε ο εξυπηρετητής NodeJS με σκοπό τη διασύνδεση του clients τμήματος της εφαρμογής (frontend) με τη βάση δεδομένων PostgreSQL. Ο εξυπηρετητής θα λαμβάνει τα αιτήματα από τον client στην πόρτα 3000. Η αποστολή των αιτημάτων από το frontend τμήμα της εφαρμογής προς τον server, θα γίνεται με χρήση της μεθόδου get. Στη συνέχεια αυτού του κεφαλαίου, θα δούμε αναλυτικά όλα τα route paths που δημιουργήσαμε στον εξυπηρετητή (αρχείο app.js) ώστε να επικοινωνεί με τη βάση και να αντλεί τα δεδομένα που χρειάζεται το frontend τμήμα της εφαρμογής.

4.4.2 RoutePath/ currentPreviousMeasurementsTank1

Αρχικά, δημιουργήσαμε το route path '/CurrentPreviousMeasurementsTank1'. Μέσω αυτού του query, θέλουμε να βρούμε τις τιμές Water_Volume και Water_Rate που σχετίζονται με το τελευταίο και το προτελευταίο timestamp που έχει καταχωρηθεί στον πίνακα Gastouri της βάσης δεδομένων μας από τη χρονική στιγμή που κάνουμε το ερώτημα. Ο εξυπηρετητής λαμβάνει ένα query με τη μέθοδο get και δημιουργεί το παρακάτω ερώτημα:

“SELECT A.WATER_VOLUME, A.TIMESTAMP AT TIME ZONE 'UTC' AT TIME ZONE 'Europe/Bucharest', A.WATER_RATE FROM GASTOURI AS A WHERE A.TIMESTAMP<=\$1”

Κεφάλαιο 4 : Υλοποίηση της εφαρμογής

Ακολούθως, ο εξυπηρετητής NodeJS, αποστέλλει το ερώτημα στη βάση δεδομένων PostgreSQL. Η επικοινωνία της βάσης δεδομένων με τον NodeJS, γίνεται μέσω της προεπιλεγμένης πόρτας 5432. Στο επόμενο βήμα γίνεται αναζήτηση στον πίνακα Gastouri των εγγραφών που έχουν μικρότερο ή ίσο timestamp από το timestamp του ερωτήματος. Η απάντηση επιστρέφει στο εξυπηρετητή NodeJS. Για να μην επιστρέψουν όλες οι τιμές του Water_Volume και Water_Rate που περιέχουν μικρότερο timestamp από αυτό που έχει ζητηθεί, Χρησιμοποιείται η μέθοδος slice με όρισμα -2 ώστε να επιστρέφουν μόνο οι 2 τελευταίες. Ο εξυπηρετητής επιστρέφει την απάντηση σε μορφή JSON. Η απάντηση είναι ενσωματωμένη σε πίνακα (array) στην ιδιότητα 'results'. Όταν το πεδίο 'status' παίρνει την τιμή 'ok', υποδηλώνει ότι η αναζήτηση ολοκληρώθηκε με επιτυχία. Αν η αναζήτηση δεν ολοκληρωθεί με επιτυχία τότε στο JSON επιστρέφεται στο πεδίο 'status' η τιμή 'not found'.

Παρακάτω, βλέπουμε το JSON response που λαμβάνει η εφαρμογή, αρχικά σε περίπτωση επιτυχούς αναζήτησης. Δηλαδή στην περίπτωση που υπάρχει εγγραφή στη βάση δεδομένων για το timestamp που αναζητήσαμε.

```
{  "status": "ok",

  "results": [{ "water_volume": 92.5995102525,

                "timezone": "2022-07-25T23:09:01.000Z",

                "water_rate": 3500},

              { "water_volume": 142.632092297,

                "timezone": "2022-07-25T23:10:01.000Z",

                "water_rate": 3500}

]
```

Ακολουθεί το JSON response που λαμβάνει η εφαρμογή σε περίπτωση που το query, περιέχει ένα timestamp με ημερομηνία που δεν αντιστοιχεί σε κάποια εγγραφή στη βάση.

```
{  "status": "notfound",

  "searchTerm": "2013-08-13 00:00:00"
```

}

Τέλος, ακολουθεί το URL μέσω του οποίου αποστέλλεται ως query, το προς αναζήτηση timestamp (μέθοδος get).

<http://localhost:3000/currentPreviousMeasurementsTank1?q=2022-07-25%2023:09:01>

4.4.3 RoutePath/currentPreviousMeasurementsTank2

Το δεύτερο route path που δημιουργήθηκε είναι το '/CurrentPreviousMeasurementsTank2 ' και αφορά τη δεξαμενή Kavos. Μέσω αυτού του query, θέλουμε να βρούμε τις τιμές Water_Volume και Water_Rate που σχετίζονται με το τελευταίο και το προτελευταίο timestamp που έχει καταχωρηθεί στον πίνακα Kavos της βάσης δεδομένων από τη χρονική στιγμή που γίνεται το ερώτημα. Ο εξυπηρετητής λαμβάνει ένα query με τη μέθοδο get, όπως αναφέρθηκε και πιο πάνω, και δημιουργεί το παρακάτω ερώτημα:

“SELECT A.WATER_VOLUME, A.TIMESTAMP AT TIME ZONE 'UTC' AT TIME ZONE 'Europe/Bucharest', A.WATER_RATE FROM KAVOS AS A WHERE A.TIMESTAMP<=\$1”

Ο εξυπηρετητής επιστρέφει την απάντηση σε μορφή JSON και τα δεδομένα βρίσκονται ενσωματωμένα στην ιδιότητα 'results' όπως και στην περίπτωση του query της δεξαμενής Gastouri.

4.4.4 RoutePath/ consumptionRegistration

Μέσω των RoutePaths CurrentPreviousMeasurementsTank1 και CurrentPreviousMeasurementsTank2 έγινε λήψη της τελευταίας τιμής του Water_Rate για την κάθε δεξαμενή καθώς και των δύο τελευταίων τιμών του Water_Volume. Με βάση αυτές τις τιμές, θα γίνει ο υπολογισμός των καταναλώσεων. Για να είναι εφικτός ο υπολογισμός της κατανάλωσης ανά λεπτό, το Water Rate μετατρέπεται από λίτρα/λεπτό σε κυβικά μέτρα. Ακολουθώντας, προστίθεται η προτελευταία τιμή Water volume με το τελευταίο water rate και στο επόμενο βήμα, από αυτό το άθροισμα αφαιρείται η τελευταία τιμή που έχει ληφθεί για το Water volume. Με αυτό τον τρόπο, δημιουργήθηκε ο κατάλληλος αλγόριθμος (αρχείο functions.js) για τον υπολογισμό της κατανάλωσης σε κάθε δεξαμενή.

Κεφάλαιο 4 : Υλοποίηση της εφαρμογής

Μέσω της εντολής Fetch, γίνεται ένα αίτημα HTTP στο NodeJS. Ο NodeJS αναλαμβάνει, μέσω του routepath '/consumptionRegistration', να εισάγει στον πίνακα statistics, τις καταναλώσεις τόσο για την δεξαμενή Gastouri όσο και για την δεξαμενή Kavos για κάθε χρονοσφραγίδα. Το query έχει την ακόλουθη μορφή:

```
"INSERT INTO STATISTICS (TIMESTAMP, CONSUMPTION_GASTOURI, CONSUMPTION_KAVOS) VALUES ($1,$2,$3)"
```

Σε περίπτωση επιτυχούς εγγραφής στη βάση δεδομένων, το JSON response που λαμβάνει η εφαρμογή, έχει στο πεδίο 'status', την τιμή 'ok'. Αν η εισαγωγή των δεδομένων στη βάση, δεν ολοκληρωθεί με επιτυχία, τότε στο JSON επιστρέφεται στο πεδίο 'status' η τιμή 'not found'. Ακολουθεί το JSON response που λαμβάνει η εφαρμογή σε περίπτωση επιτυχούς εγγραφής:

```
{      "status": "ok"
}
```

Παρακάτω, φαίνεται ένα ενδεικτικό URL μέσω του οποίου αποστέλλονται ως query με χρονοσφραγίδα 25/07/2022 21:04:01, οι καταναλώσεις ανά λεπτό για τις δεξαμενές.

<http://localhost:3000/consumptionRegistration?q1=%222022-07-25%2021:04:01%22&q2=2.596009999999999926&q3=9.5502219999>

4.4.5 RoutePath / statisticsHourlyTank1

Το επόμενο route path, είναι το '/statisticsHourlyTank1'. Μέσω αυτού του query, επιτυγχάνεται η εύρεση της μέσης τιμής της κατανάλωσης (mean Consumption) σε διάστημα μίας ώρας για την δεξαμενή Gastouri από τον πίνακα statistics. Όπως έχει αναφερθεί και νωρίτερα, στον πίνακα statistics έχει εισαχθεί η κατανάλωση ανά λεπτό με τη βοήθεια του routePath 'consumptionRegistration'. Μέσω της συνάρτησης calculateHourlyConsumption που έχει δημιουργηθεί στο αρχείο d3form.js, καθορίζεται η αρχή και το τέλος του χρονικού διαστήματος που πρέπει να αναζητήσει ο NodeJS (timestampStart & timestampEnd) στον πίνακα statistics της βάσης δεδομένων. Στη συνέχεια, έχοντας δημιουργήσει έναν βρόγχο ο οποίος εκτελείται τόσες φορές όσες και το μήκος των γραμμών της απάντησης, αθροίζονται οι

Κεφάλαιο 4 : Υλοποίηση της εφαρμογής

καταναλώσεις και ακολούθως διαιρούνται με το πλήθος των τιμών ώστε να υπολογιστεί η κατανάλωση ανά ώρα. Ο εξυπηρετητής λαμβάνει ένα query με τη μέθοδο get και δημιουργεί το παρακάτω ερώτημα:

```
“SELECT A.CONSUMPTION_GASTOURI,A.TIMESTAMP AT TIME ZONE 'UTC' AT TIME ZONE  
'Europe/Bucharest' FROM STATISTICS AS A WHERE TIMESTAMP>=$1 AND TIMESTAMP<=$2”
```

Η απάντηση είναι ενσωματωμένη στην ιδιότητα ‘results’ ενώ το πεδίο ‘status’ παίρνει την τιμή ‘ok’ που υποδηλώνει ότι η αναζήτηση ολοκληρώθηκε με επιτυχία. Αν η αναζήτηση δεν ολοκληρωθεί με επιτυχία, τότε στο JSON επιστρέφεται στο πεδίο ‘status’ η τιμή ‘not found’. Παρακάτω, βλέπουμε το αντίστοιχο JSON response που λαμβάνει η εφαρμογή σε περίπτωση επιτυχούς αναζήτησης. Στο πεδίο consumption, αντιστοιχεί η μέση τιμή κατανάλωσης ανά λεπτό. Στο πεδίο timestamp, μας επιστρέφεται η χρονοσφραγίδα από την οποία ξεκινάει το χρονικό διάστημα το οποίο αναζητήθηκε:

```
{      "status":"ok",  
  
      "results":{"timestamp":"2022-04-25T19:43:01",  
  
                "consumption":10.048122389830503}  
}
```

Τέλος, ακολουθεί το URL μέσω του οποίου αποστέλλονται ως query, τα timestamps του χρονικού διαστήματος που θέλει να αναζητήσει ο χρήστης.

<http://localhost:3000/statisticsHourlyTank1?q1=%222022-04-25%2019:43:00%22&q2=%222022-04-25%2020:43:00%22>

4.4.6 RoutePath / statisticsDailyTank1

Το επόμενο route path, είναι το '/statisticsDailyTank1'. Μέσω αυτού του query, επιτυγχάνεται η εύρεση της μέσης τιμής της κατανάλωσης (mean Consumption) κάθε ημέρας για τη δεξαμενή Gastouri από τον πίνακα statistics. Όπως και στο route path '/statisticsHourlyTank1' έτσι και εδώ, γίνεται χρήση της συνάρτησης calculateHourlyConsumption που έχει δημιουργηθεί στο αρχείο d3form.js. Η διαφοροποίηση είναι ότι έχει καθοριστεί με διαφορετικό τρόπο η αρχή και το τέλος του χρονικού διαστήματος που θα αναζητήσει ο NodeJS (timestampStart & timestampEnd) στον πίνακα statistics της βάσης δεδομένων. Στην περίπτωση του '/statisticsHourlyTank1', η αρχική χρονοσφραγίδα με την τελική, είχαν διαφορά μία

Κεφάλαιο 4 : Υλοποίηση της εφαρμογής

ώρα. Στην περίπτωση του daily, η αρχική με την τελική χρονοσφραγίδα, έχουν διαφορά είκοσι τέσσερις ώρες. Στη συνέχεια, έχοντας δημιουργήσει έναν βρόγχο ο οποίος εκτελείται τόσες φορές όσες και το μήκος των γραμμών της απάντησης, αθροίζονται οι καταναλώσεις και στη συνέχεια διαιρούνται με το πλήθος των τιμών για τον υπολογισμό της κατανάλωσης ανά ημέρα. Ο εξυπηρετητής λαμβάνει ένα query με τη μέθοδο get και δημιουργεί το παρακάτω ερώτημα:

```
“SELECT A.CONSUMPTION_GASTOURI,A.TIMESTAMP AT TIME ZONE 'UTC' AT TIME ZONE  
'Europe/Bucharest' FROM STATISTICS AS A WHERE TIMESTAMP>=$1 AND TIMESTAMP<=$2”
```

Ακολουθεί το αντίστοιχο JSON response που λαμβάνει η εφαρμογή σε περίπτωση επιτυχούς αναζήτησης.

```
{      "status": "ok",  
  
      "results": {"timestamp": "2022-04-27T00:01:01.000Z",  
  
                 "consumption": 10.261818674911705}  
  
}
```

Τέλος, το URL μέσω του οποίου αποστέλλονται ως query τα timestamps του χρονικού διαστήματος που θέλει να αναζητήσει ο χρήστης, είναι:

<http://localhost:3000/statisticsDailyTank1?q1=%222022-04-27%2000:00:00%22&q2=%222022-04-27%2023:59:00%22>

4.4.7 RoutePath / statisticsHourlyTank2

Το επόμενο route path που δημιουργήθηκε είναι το '/statisticsHourlyTank2' και έχει σχεδιαστεί με την ίδια ακριβώς λογική όπως και το αντίστοιχο του '/statisticsHourlyTank1'. Μέσω αυτού του query, επιτυγχάνεται η εύρεση της μέσης τιμής της κατανάλωσης (mean Consumption) σε διάστημα μίας ώρας για τη δεξαμενή Kavos από τον πίνακα statistics. Ο εξυπηρετητής λαμβάνει ένα query με τη μέθοδο get και δημιουργεί ένα αντίστοιχο ερώτημα, αυτήν τη φορά για τη δεξαμενή Kavos. Το ερώτημα έχει την παρακάτω μορφή:

```
“SELECT A.CONSUMPTION_KAVOS,A.TIMESTAMP AT TIME ZONE 'UTC' AT TIME ZONE  
'Europe/Bucharest' FROM STATISTICS AS A WHERE TIMESTAMP>=$1 AND TIMESTAMP<=$2”
```

Κεφάλαιο 4 : Υλοποίηση της εφαρμογής

Αντίστοιχη με την απάντηση στο query statisticsHourly Tank1, είναι και η απάντηση στο query statisticsHourly Tank2. Ακολουθεί το response που λαμβάνει η εφαρμογή σε περίπτωση επιτυχούς αναζήτησης .

```
{      "status":"ok",

      "results":{"timestamp":"2022-04-27T21:50:01.000Z",

      "consumption":3.7053008474575613}}
```

Τέλος, ακολουθεί το URL μέσω του οποίου αποστέλλονται ως query, τα timestamps του διαστήματος που θέλει ο χρήστης να αναζητήσει.

<http://localhost:3000/statisticsHourlyTank2?q1=%222022-04-27%2021:50:00%22&q2=%222022-04-27%2022:50:00%22>

4.4.8 RoutePath / statisticsDailyTank2

Το τελευταίο route path, είναι το '/statisticsDailyTank2'. Έχει σχεδιαστεί με την ίδια ακριβώς λογική όπως και το αντίστοιχο του '/statisticsDailyTank1'. Το ερώτημα έχει την παρακάτω μορφή:

“SELECT A.CONSUMPTION_KAVOS,A.TIMESTAMP AT TIME ZONE 'UTC' AT TIME ZONE 'Europe/Bucharest' FROM STATISTICS AS A WHERE TIMESTAMP>=\$1 AND TIMESTAMP<=\$2”

Το JSON response που λαμβάνει η εφαρμογή σε περίπτωση επιτυχούς αναζήτησης, δε διαφοροποιείται καθόλου από αυτό του statisticsHourly Tank1. Τέλος, το URL μέσω του οποίου αποστέλλονται ως query τα timestamps του χρονικού διαστήματος που θέλει ο χρήστης να αναζητήσει είναι:

<http://localhost:3000/statisticsDailyTank2?q1=%222022-04-28%2000:00:00%22&q2=%222022-04-28%2023:59:00%22>

4.5 Ανάπτυξη του τμήματος της εφαρμογής που αλληλεπιδρά με τον χρήστη (Front-End)

Το client side τμήμα της εφαρμογής έχει να κάνει με το γραφικό περιβάλλον(GUI) το οποίο αλληλεπιδρά με τον χρήστη. Ως προς τον σχεδιασμό του, αποτελείται από τα αρχεία index.html, d3form.js, drawtank.js, main.js, functions.js, time.js, tooltip.js, language.js καθώς και το αρχείο style.css. Το αρχείο index.html είναι υπεύθυνο για την εμφάνιση της html σελίδας στον φυλλομετρητή.

Στο τμήμα της κεφαλίδας του αρχείου index.html, φορτώνονται οι απαραίτητες βιβλιοθήκες για τη λειτουργία της εφαρμογής. Αρχικά, εισάγετε η βιβλιοθήκη jquery που είναι απαραίτητη για την σύνταξη των jqueryies που χρησιμοποιούνται σε μεγάλο βαθμό στον κώδικα. Μία ακόμη βιβλιοθήκη είναι η bootstrap μέσω την οποίας φορτώνονται στοιχεία που θα χρησιμοποιηθούν στο μενού περιήγησης της εφαρμογής. Επίσης, η D3 αποτελεί μια βιβλιοθήκη της JavaScript πάνω στην οποία έχει βασιστεί η δημιουργία των στατιστικών και τέλος, στην κεφαλίδα του html, φορτώνεται και το αρχείο style.css που είναι υπεύθυνο για τη γραφική απεικόνιση.

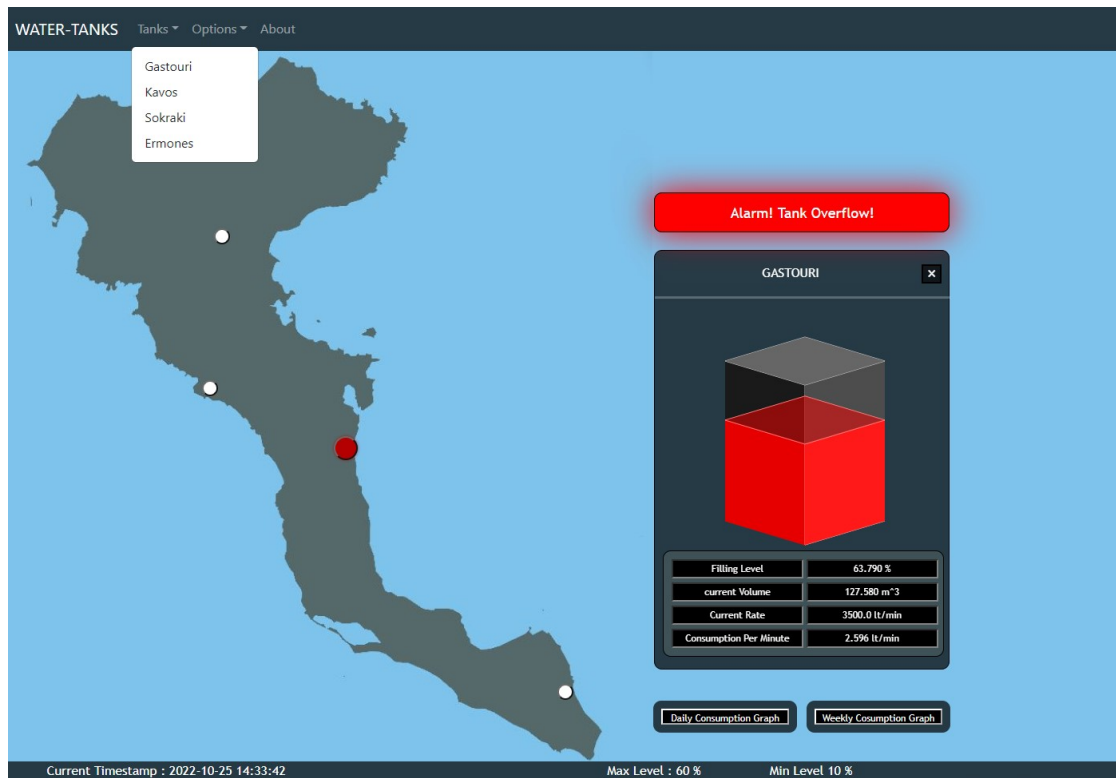
Στο σώμα του html αρχείου, βρίσκεται το μενού της εφαρμογής το οποίο έχει υλοποιηθεί μέσω του bootstrap navigation bar. Ακόμη, υπάρχουν όλα τα απαραίτητα τμήματα (divs) και φόρμες (forms) της εφαρμογής ενώ στο κάτω μέρος της αρχικής οθόνης υπάρχει ένα status bar.

4.5.1 Μενού περιήγησης

Το μενού περιήγησης εμφανίζεται στην κορυφή της οθόνης και περιλαμβάνει τις επιλογές tanks, options και about. Η επιλογή tanks αποτελείται από ένα drop down μενού το οποίο ξεδιπλώνεται κάνοντας κλικ. Στη συνέχεια, από το μενού, μπορεί να επιλεγεί μια από τις 4 δεξαμενές του συστήματος.

Η επόμενη επιλογή του μενού είναι η option. Με κλικ στο option, εμφανίζεται ένα drop down μενού από όπου τροποποιούνται τα επίπεδα ενεργοποίησης ενός alarm και γίνεται αλλαγή της γλώσσας εμφάνισης της εφαρμογής. Μέσω της επιλογής set language Greek, γίνεται εναλλαγή από Αγγλικά σε Ελληνικά και το αντίστροφο. Η τελευταία επιλογή στο μενού, είναι η about. Επιλέγοντας το about εμφανίζονται στοιχεία που αφορούν την πτυχιακή εργασία, αποτέλεσμα της οποίας αποτελεί αυτή η εφαρμογή. Λεπτομέρειες σχετικά με την επιλογή γλώσσας και την επιλογή about αναλύονται στη συνέχεια.

Κεφάλαιο 4 : Υλοποίηση της εφαρμογής



Εικόνα 5: Μενού περιήγησης

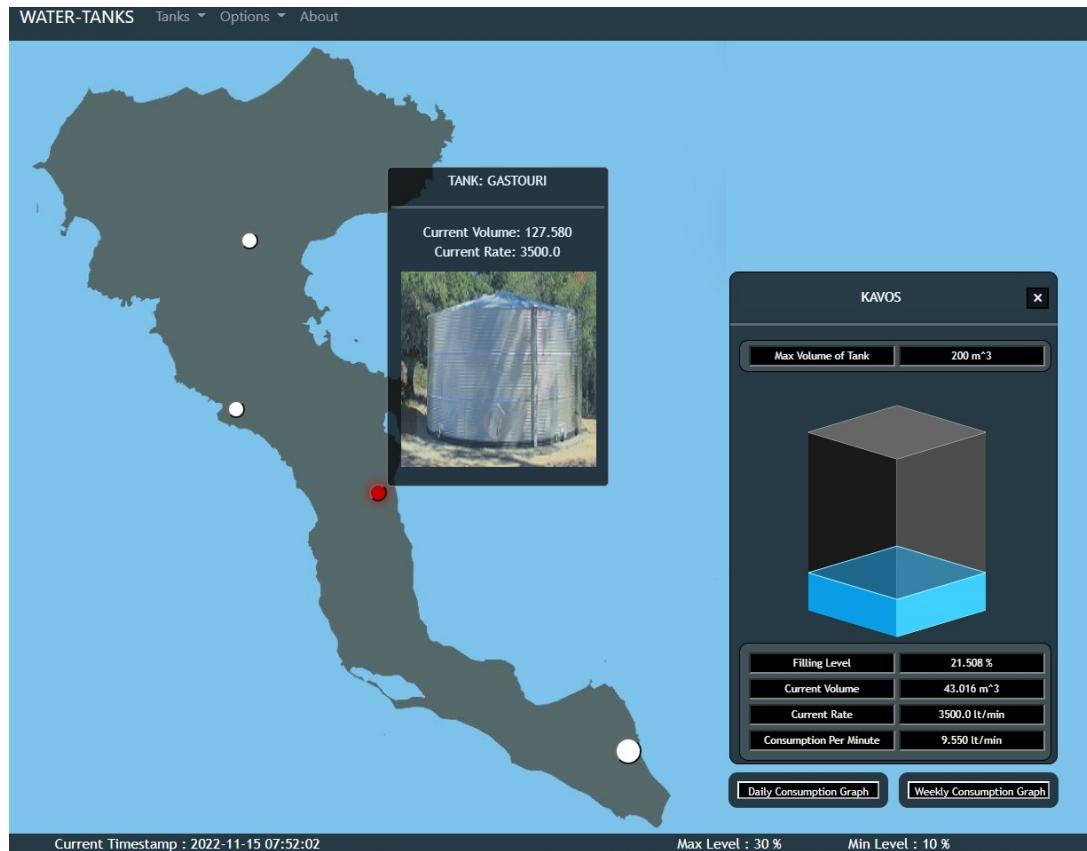
4.5.2 Χάρτης

Όταν ο χρήστης ανοίξει την εφαρμογή Water Tanks στην αρχική οθόνη εμφανίζεται ένας χάρτης της Κέρκυρας. Πάνω στον χάρτη εμφανίζονται οι τοποθεσίες που είναι τοποθετημένες οι δεξαμενές. Κάθε δεξαμενή αναπαριστάται με μια άσπρη τελεία πάνω στον χάρτη. Κάνοντας hover με το ποντίκι πάνω σε μια δεξαμενή, ο χρήστης μπορεί να δει μέσω ενός συμβουλευτικού πίνακα που εμφανίζεται (tool tip), πληροφορίες σχετικές με τη δεξαμενή. Οι πληροφορίες που εμφανίζονται στο tool tip πίνακα, αφορούν το όνομα της δεξαμενής, φωτογραφία της δεξαμενής, τον τρέχοντα όγκο του νερού μέσα στη δεξαμενή καθώς και την τρέχουσα τιμή του νερού που εισέρχεται στη δεξαμενή μέσω της αντλίας.

Σε περίπτωση που ο όγκος του νερού μέσα στη δεξαμενή, ξεπεράσει το ανώτερο ή κατώτερο επιτρεπτό επίπεδο, τότε ενεργοποιείται ο μηχανισμός διέγερσης ο οποίος έχει σχεδιαστεί για να ενημερώνει τον χρήστη. Τότε, το χρώμα της τελείας στη δεξαμενή που έχει ενεργοποιηθεί ο συναγερμός, αλλάζει και από λευκό γίνεται κόκκινο και ενεργοποιείται ο ηχητικός συναγερμός. Με αυτόν τον τρόπο, ο χρήστης δε χρειάζεται να ελέγχει συνέχεια τις δεξαμενές του συστήματος για την καλή τους λειτουργία. Σε περίπτωση άρσης της αιτίας που προκάλεσε τον συναγερμό, η τελεία που αναπαριστά τη δεξαμενή

Κεφάλαιο 4 : Υλοποίηση της εφαρμογής

επιστρέφει στο αρχικό της χρώμα. Τέλος, για να μπορεί να συσχετίζει ο χρήστης την επιλογή δεξαμενής που έχει κάνει με την τοποθεσία της στον χάρτη, η ενδεικτική τελεία της δεξαμενής που έχει επιλεγεί, γίνεται διπλάσια σε μέγεθος.



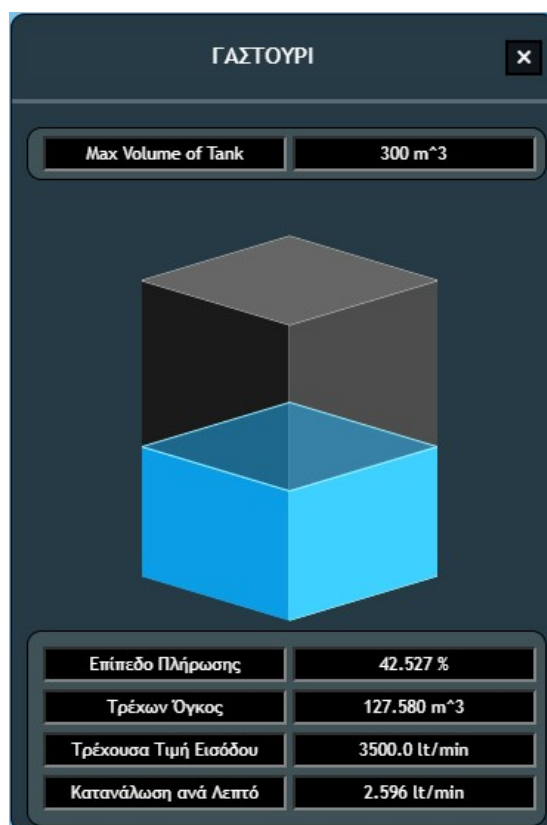
Εικόνα 6: Βασικός χάρτης εφαρμογής

4.5.3 Εμφάνιση δεξαμενών

Η επιλογή μιας δεξαμενής, μπορεί να γίνει με δύο τρόπους. Μέσω του μενού Tanks ή κάνοντας κλικ πάνω στον χάρτη στην δεξαμενή που μας ενδιαφέρει. Μετά την επιλογή της δεξαμενής, εμφανίζεται ένα παράθυρο με τη γραφική αναπαράσταση την δεξαμενής. Στην κεφαλίδα του παραθύρου, υπάρχει το όνομα της δεξαμενής. Ακριβώς από κάτω, αναγράφεται η μέγιστη χωρητικότητα νερού που μπορεί να χωρέσει στη δεξαμενή, μετρούμενη σε κυβικά μέτρα (η μέγιστη χωρητικότητα είναι διαφορετική σε κάθε δεξαμενή). Εκτός από την πληρότητα της δεξαμενής που αναπαριστάται με γραφικό τρόπο, κάτω από τη δεξαμενή υπάρχουν πληροφορίες σχετικές με αυτήν. Αρχικά, εμφανίζεται το ποσοστό πληρότητας της δεξαμενής (Filling level). Ακριβώς από κάτω, βλέπουμε τον τρέχοντα όγκο νερού (current Volume) που

Κεφάλαιο 4 : Υλοποίηση της εφαρμογής

υπάρχει στη δεξαμενή, μετρούμενο σε κυβικά μέτρα m^3 . Το επόμενο πεδίο, αφορά την τρέχουσα τιμή της ποσότητας του νερού που εισέρχεται στη δεξαμενή από την αντλία όταν αυτή βρίσκεται σε λειτουργία (Current Rate). Τέλος, έχουμε μια μέτρηση σχετική με την κατανάλωση του νερού της δεξαμενής το τελευταίο λεπτό. Σε αυτό το σημείο, πρέπει να αναφερθεί ότι όλες οι πληροφορίες που λαμβάνονται από τη δεξαμενή, ανανεώνονται κάθε ένα λεπτό καθώς οι αισθητήρες που είναι τοποθετημένοι στη δεξαμενή, παίρνουν νέα μέτρηση ανά λεπτό.

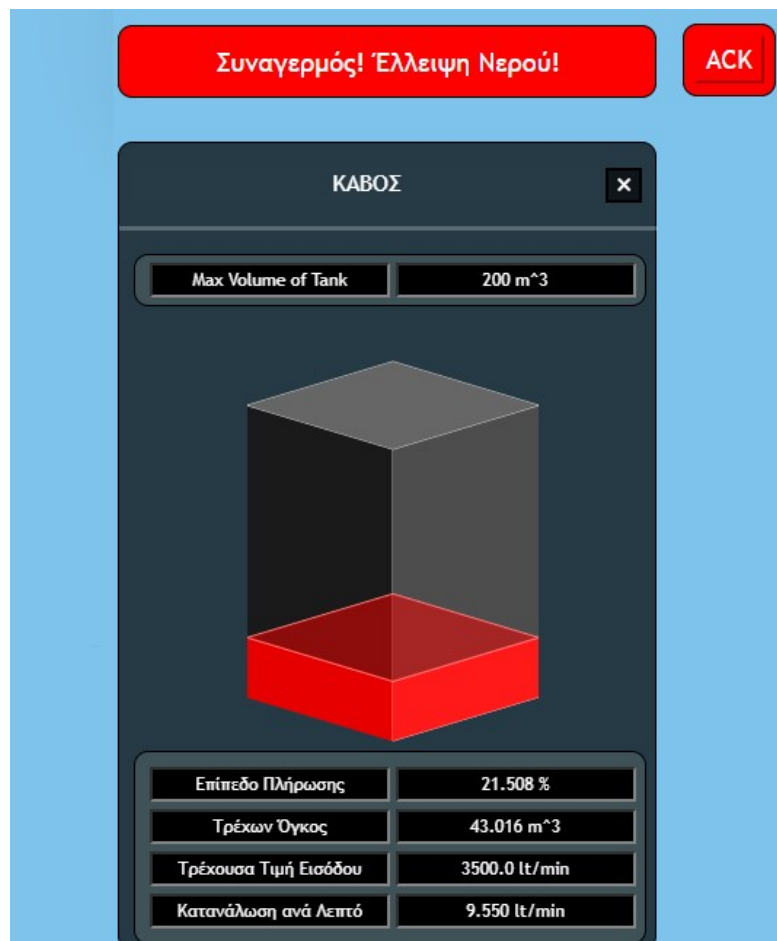


Εικόνα 7: Γραφική αναπαράσταση δεξαμενής

Σε περίπτωση ενεργοποίησης του συναγερμού είτε λόγω υπερχείλισης ή λόγω έλλειψης νερού, το χρώμα του νερού της δεξαμενής γίνεται κόκκινο. Επιπρόσθετα, ένα οπτικό μήνυμα alarm εμφανίζεται πάνω από τη δεξαμενή με κόκκινο χρώμα που αναβοσβήνει ώστε να τραβήξει την προσοχή του χειριστή. Στο οπτικό μήνυμα εμφανίζεται κείμενο που ενημερώνει τον χρήστη αν πρόκειται για υπερχείλιση της δεξαμενής ή έλλειψη νερού.

Εκτός από το οπτικό μήνυμα alarm υπάρχει και ηχητικό alarm. Η ενεργοποίησή του γίνεται μαζί με το οπτικό alarm. Βασικός σκοπός της ύπαρξης του ηχητικού alarm, είναι η ενημέρωση του χρήστη σε

περίπτωση που δε βρίσκεται κοντά στο τερματικό. Μετά την ενημέρωσή του, ο χρήστης μπορεί να απενεργοποιήσει το ηχητικό alarm μέσω του κουμπιού acknowledge (ACK). Ακόμη και μετά την απενεργοποίηση του ηχητικού alarm το οπτικό παραμένει για να υπενθυμίζει στον χρήστη την δυσλειτουργία στο σύστημα. Η απενεργοποίηση του οπτικού alarm γίνεται μετά την άρση του προβλήματος.



Εικόνα 8: Γραφική αναπαράσταση δεξαμενής σε κατάσταση alarm

4.5.4 Γραμμή κατάστασης

Χαμηλά στην αρχική οθόνη έχει δημιουργηθεί μια γραμμή κατάστασης status bar. Σκοπός του status bar είναι η ενημέρωση του χρήστη της εφαρμογής για την τελευταία χρονική στιγμή (timestamp) που η εφαρμογή μας άντλησε δεδομένα από τη βάση. Με αυτό τον τρόπο, ο χρήστης έχει τη δυνατότητα να αντιληφθεί την περίπτωση που υπάρχει δυσλειτουργία στην εφαρμογή. Εκτός του current timestamp, το

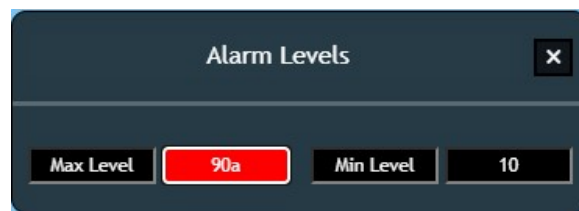
status bar περιέχει πληροφορίες σχετικά με την τελευταία τροποποίηση που έχει κάνει ο χρήστης σχετικά με το μέγιστο και ελάχιστο επιτρεπτό ποσοστό πλήρωσης.



Εικόνα 4.6: Status Bar

4.5.5 Καθορισμός επιπέδων συναγερμού

Επιλέγοντας από το μενού options→Set alarm levels, ο χρήστης μπορεί να αλλάξει τα επιτρεπτά επίπεδα συναγερμού μέσω μιας φόρμας. Στο αρχείο functions.js, κατά τη δημιουργία της συνάρτησης SetAlarmLevels, καθορίστηκε μια σειρά από κριτήρια ώστε ο χρήστης να μην μπορεί να εισάγει μη επιτρεπτές τιμές. Ο χρήστης πρέπει να εισάγει αριθμό μέσα στα επιτρεπτά όρια δηλαδή μια τιμή μεταξύ του 1 και του 99. Η εισαγωγή τόσο της ελάχιστης όσο και την μέγιστης τιμής δεν μπορεί να περιέχει χαρακτήρες. Τέλος, η μέγιστη τιμή δεν μπορεί να είναι μικρότερη της ελάχιστης. Στις περιπτώσεις που ο χρήστης δοκιμάσει να εισάγει λανθασμένη τιμή, το κελί αλλάζει χρώμα και γίνεται κόκκινο και η διαδικασία της αλλαγής δεν μπορεί να ολοκληρωθεί. Σε αυτή την περίπτωση, πέρα από το χρώμα του κελίου που αλλάζει, ο χρήστης δεν μπορεί να κλείσει το παράθυρο και να ολοκληρώσει τη διαδικασία. Ένα μήνυμα σφάλματος εμφανίζεται για να τον ενημερώσει ότι πρέπει να εισάγει μια αποδεκτή τιμή ώστε να ολοκληρωθεί η διαδικασία πριν αποχωρήσει.



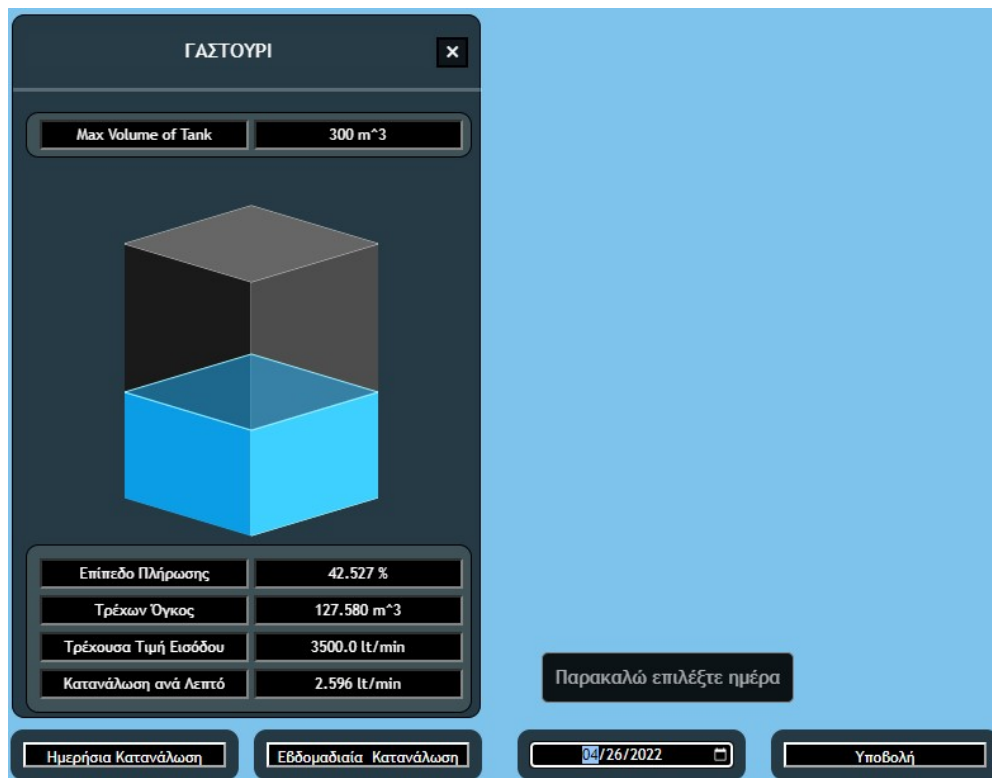
Εικόνα 9: Φόρμα εισαγωγής επιπέδων συναγερμού

4.5.6 Στατιστικά

Για τη δημιουργία των στατιστικών, έγινε χρήση της βιβλιοθήκης D3 ώστε να απεικονιστεί η μέση τιμή της κατανάλωσης ανά δεξαμενή σε ημερήσια και εβδομαδιαία βάση με τη μορφή ραβδογραμμάτων στήλης. Για την υλοποίηση των στατιστικών, έγινε χρήση του πίνακα statistics της βάσης δεδομένων, στον οποίο έχουμε καταχωρίσει την κατανάλωση ανά λεπτό για τις δεξαμενές Gastouri και Kavos.

Κεφάλαιο 4 : Υλοποίηση της εφαρμογής

Κάνοντας επιλογή δεξαμενής από την αρχική οθόνη της εφαρμογής, εμφανίζονται, εκτός της δεξαμενής, και δύο κουμπιά που σχετίζονται με τα στατιστικά της. Ακολούθως, ο χρήστης μπορεί να επιλέξει να δει τη μέση ημερήσια κατανάλωση ή τη μέση εβδομαδιαία κατανάλωση. Κάνοντας κλικ σε μία από τις δύο διαθέσιμες επιλογές, εμφανίζεται ένα κουμπί date μαζί με ένα μήνυμα που προτρέπει το χρήστη να επιλέξει την επιθυμητή ημερομηνία από το ημερολόγιο που εμφανίζεται. Στην περίπτωση επιλογής της εβδομαδιαίας κατανάλωσης, ο χρήστης επιλέγει από το ημερολόγιο την αρχική ημέρα της εβδομάδας. Η εφαρμογή, με αυτόματο τρόπο, υπολογίζει την ημέρα που ολοκληρώνεται η εβδομάδα, μέσω κριτηρίων που έχουν δημιουργηθεί στον κώδικα και σχετίζονται με χαρακτηριστικά όπως ο έλεγχος αν πρόκειται για δίσεκτη χρόνια, οι ημέρες του μήνα (30 ή 31) , η επιλογή αρχικής ημέρας της εβδομάδας που έχει σαν αποτέλεσμα την ολοκλήρωση της εβδομάδας το επόμενο ημερολογιακό έτος. Μετά την επιλογή ημερομηνίας, εμφανίζεται το κουμπί submit ώστε να είναι δυνατή η υποβολή του αιτήματος.



Εικόνα 10: κουμπιά επιλογής στατιστικών

Τροποποιώντας τον κώδικα Bar chart της βιβλιοθήκης D3, δημιουργούνται τα ραβδογράμματα που χρησιμοποιούνται στην απεικόνιση της ημερήσιας και εβδομαδιαίας κατανάλωσης. Στο ραβδόγραμμα της ημερήσιας κατανάλωσης στον άξονα x, εμφανίζεται η βαθμονόμηση της μέσης κατανάλωσης ενώ

Κεφάλαιο 4 : Υλοποίηση της εφαρμογής

στον άξονα ψ , οι ώρες της ημέρας από 0 έως 23. Αντίστοιχα, στο ραβδόγραμμα της εβδομαδιαίας κατανάλωσης στο άξονα χ , εμφανίζεται η μέση τιμή της κατανάλωσης ανά ημέρα και στο άξονα ψ , η ημερομηνία της κάθε ημέρας. Ακολουθούν ενδεικτικές φωτογραφίες.



Εικόνα 11: Ραβδογράμματα μέσης ημερήσιας και εβδομαδιαίας κατανάλωσης

4.5.7 Επιλογή γλώσσας

Επιλέγοντας από το μενού options→ set language ο χρήστης μπορεί να αλλάξει τη γλώσσα που θα εμφανίζεται στο μενού και σε όλα τα μηνύματα της εφαρμογής. Οι γλώσσες επιλογής είναι τα Αγγλικά και τα Ελληνικά. Για την επιλογή γλώσσας, έχουν δημιουργηθεί μέσα στο αρχείο language.js, εμφωλευμένοι πίνακες δύο στοιχείων μέσα στον αρχικό πίνακα. Στο στοιχείο μηδέν του κάθε πίνακα, βρίσκεται το αγγλικό κείμενο και στο στοιχείο ένα το ελληνικό κείμενο. Ανάλογα με την επιλογή γλώσσας που κάνουμε, εμφανίζεται και το αντίστοιχο κείμενο.

4.5.8 About

Στην επιλογή about του μενού περιήγησης, έχει δημιουργηθεί μια φόρμα με τον τίτλο της πτυχιακής εργασίας, αποτέλεσμα της οποίας είναι η δημιουργία της εφαρμογής Water Tanks. Επιπλέον,

αναγράφονται τα ονόματα του επιβλέποντα καθηγητή και του φοιτητή. Ακολουθεί ενδεικτικό screen shot.



Εικόνα 12: Φόρμα about

4.6 Δεδομένα

Τα δεδομένα που χρησιμοποιήθηκαν για τη δημιουργία την βάσης όσον αφορά την δεξαμενή Gastouri, προέρχονται από το εργαστήριο CMODLab του τμήματος πληροφορικής του Ιονίου Πανεπιστημίου. Πιο συγκεκριμένα, προέρχονται από αισθητήρες που έχει τοποθετήσει το εργαστήριο στη δεξαμενή. Η πρώτη τιμή προέρχεται από την αντλία της δεξαμενής του Γαστουρίου και αφορά τη μέση τιμή του νερού που εισέρχεται στη δεξαμενή μετρούμενη σε λίτρα ανά λεπτό. Η δεύτερη τιμή αφορά τον αισθητήρα που είναι τοποθετημένος στην κορυφή της δεξαμενής του Γαστουρίου και μετράει την απόσταση του αισθητήρα από την επιφάνεια του νερού. Στη συνέχεια, με χρήση κατάλληλου αλγόριθμου, μετατρέπει την τιμή αυτή σε κυβικά μέτρα (m^3) ώστε να γνωρίζουμε τον όγκο του νερού που είναι αποθηκευμένο στη δεξαμενή.

Τα δεδομένα που αφορούν τη δεξαμενή Κάβος δεν είναι πραγματικά. Η δημιουργία τους έγινε τεχνητά με βάση τις τιμές από την δεξαμενή Γαστούρι με σκοπό τον έλεγχο της καλής λειτουργίας του κώδικα.

4.7 Εγκατάσταση λογισμικού εφαρμογής

Σε αυτήν την παράγραφο, θα γίνει μια σύντομη παρουσίαση της διαδικασίας εγκατάστασης του λογισμικού που είναι απαραίτητο για τη σωστή λειτουργία της εφαρμογής. Το λειτουργικό σύστημα στο οποίο τρέχει η εφαρμογή Water Tanks είναι τα Windows 10 Education 64-bit. Το υλικό (hardware) στο οποίο τρέχει η εφαρμογή, είναι ένα laptop Lenovo με επεξεργαστή Intel(R) Core(TM) i5-7200U CPU 2.50GHz και μνήμη RAM 8GB. Το λογισμικό που εγκαταστήθηκε στο πλαίσιο της εργασίας, είναι η PostgreSQL για τη δημιουργία της βάσης δεδομένων και ο NodeJS για το Backend τμήμα της εφαρμογής. Τέλος, πρέπει να αναφερθεί ότι για τη σωστή λειτουργία την εφαρμογής, απαιτείται ανάλυση οθόνης 1920x1080.

4.7.1 Βάση Δεδομένων PostgreSQL

Αρχικά, μέσα από την επίσημη ιστοσελίδα της PostgreSQL και με βάση το διαθέσιμο λειτουργικό σύστημα, επιλέχθηκε η λήψη της κατάλληλης έκδοσης (PostgreSQL 14.4).

Ξεκινώντας την εγκατάσταση έγινε επιλογή των επιμέρους προγραμμάτων που θα εγκατασταθούν (PgAdmin, StackBuilder). Πριν από την ολοκλήρωση της εγκατάστασης, ο installer θα ζητήσει τη δημιουργία ενός λογαριασμού (super user) μαζί με το κατάλληλο συνθηματικό ώστε η πρόσβαση στη βάση να γίνεται μόνο από εξουσιοδοτημένα πρόσωπα.

Μετά την ολοκλήρωση της εγκατάστασης, η εφαρμογή είναι έτοιμη για χρήση. Στη συνέχεια, μέσω του γραφικού περιβάλλοντος διαχείρισης pgAdmin, μπορούν να γίνουν οι πρώτες βασικές ρυθμίσεις στην εφαρμογή. Επιλέγοντας create Server, δημιουργείται ο εξυπηρετητής της βάσης και στη συνέχεια εισάγεται το όνομα και επιλέγεται η πόρτα επικοινωνίας, που για την Water Tanks είναι το port 5432.

4.7.2 Εξυπηρετητής NodeJS

Αρχικά, επιλέχθηκε η επιθυμητή έκδοση εγκατάστασης μέσω της ιστοσελίδας <https://nodejs.org/en/>. Για την υλοποίηση της εφαρμογής, γίνεται χρήση της έκδοσης 16.16.0. Μετά την εκτέλεση του installer, με χρήση του command line των Windows, εκτελείται η εντολή

```
powershell -Command "Start-Process cmd -Verb RunAs"
```

Κεφάλαιο 4 : Υλοποίηση της εφαρμογής

Στη συνέχεια, στο παράθυρο του powershell, εκτελείται η εντολή

```
npm install -g windows-build-tools
```

Η εκκίνηση του εξυπηρετητή NodeJS, γίνεται μέσω της εντολής node και του αρχείου javascript στο οποίο υπάρχει ο κώδικας του server. Στην εφαρμογή Water Tanks, πρόκειται για το αρχείο app.js

```
Node app.js
```

Με σκοπό την αυτοματοποίηση της διαδικασίας εκκίνησης του server, δημιουργήθηκε ένα αρχείο .bat μέσω του οποίου εκτελούνται οι εντολές

```
cd src
```

```
node app.js
```

4.8 Προβλήματα κατά την υλοποίηση της εφαρμογής

Κατά τη διάρκεια υλοποίησης της εφαρμογής αντιμετωπίστηκαν διάφορα προβλήματα. Τα πιο πολλά από αυτά επιλύθηκαν σχετικά εύκολα κάποια άλλα ήταν πολύπλοκα και η επίλυση τους έγινε μέσω διαφοροποίησης του τρόπου υλοποίησης της εφαρμογής.

Ένα τέτοιο πρόβλημα αφορούσε στη δημιουργία των στατιστικών. Κατά τον υπολογισμό της ημερήσιας και της εβδομαδιαίας μέσης κατανάλωσης, στέλνονται στη βάση δεδομένων πολλαπλά αιτήματα. Συγκεκριμένα, για τον υπολογισμό της ημερήσιας κατανάλωσης, πραγματοποιούνται 24 ερωτήματα, ένα για κάθε ώρα της ημέρας. Για τον υπολογισμό της εβδομαδιαίας κατανάλωσης, αποστέλλονται 7 ερωτήματα, ένα για κάθε ημέρα της εβδομάδας. Λόγω όμως της ασύγχρονης λειτουργίας του NodeJS κατά την αποστολή ερωτημάτων στη βάση, είναι πιθανό η λήψη των αποτελεσμάτων των καταναλώσεων να γίνει με μη γραμμικό τρόπο. Το πρόβλημα αυτό λύθηκε με χρήση μιας συνάρτησης callback μέσω της οποίας διασφαλίζεται η λήψη των καταναλώσεων με την σειρά που έχουν ζητηθεί.

Ένα ακόμη πρόβλημα που σχετιζόταν με την δημιουργία των στατιστικών αφορούσε στη σχεδίαση των γραφημάτων. Μερικές φορές δεν εμφανίζονταν όλες οι τιμές στο ραβδόγραμμα. Αρχικά, έγινε προσπάθεια επίλυσης του προβλήματός με την εισαγωγή χρονοκαθυστερήσης (delay) ώστε να προλαβαίνει ο εξυπηρετητής να φέρει όλες τις τιμές πριν σχεδιαστεί το γράφημα. Αυτός ο τρόπος όμως, εμφάνιζε αρκετά μεγάλη καθυστέρηση στην απεικόνιση των αποτελεσμάτων πάνω στη φόρμα του γραφήματος. Η λύση δόθηκε, τελικά, με τη δημιουργία ενός βρόχου μέσω του οποίου γίνεται σάρωση

Κεφάλαιο 4 : Υλοποίηση της εφαρμογής

του πίνακα που περιέχει τις καταναλώσεις (consumption array) μέχρι να επιβεβαιωθεί ότι υπάρχουν δεδομένα σε όλες τις θέσεις του πίνακα. Όταν ολοκληρωθεί ο έλεγχος τότε η εφαρμογή προχωρεί στη σχεδίαση του γραφήματος.

Ένα ακόμη πρόβλημα αφορούσε την λανθασμένη τιμή που είχε η ώρα της χρονοσφραγίδας κατά την εισαγωγή των καταναλώσεων στον πίνακα statistic της βάσης. Η χρονοσφραγίδα που αποστέλλονταν μαζί με τα δεδομένα από τα ερωτήματα CurrentPreviousMeasurements Tank1 και CurrentPreviousMeasurements Tank2, επέστρεφε στον NodeJS σε ώρα UTC. Τα δεδομένα από τις στήλες current volume και current rate, όπως έχει ήδη αναφερθεί, χρησιμοποιούνται για τον υπολογισμό των καταναλώσεων. Κατά την εισαγωγή των καταναλώσεων στον πίνακα statistics στην στήλη timestamp, η χρονοσφραγίδα ήταν σε ώρα UTC. Η απόκλιση μεταξύ αυτών των χρονοσφραγίδων του πίνακα statistics και των πινάκων των δεξαμενών, στους οποίους η ώρα ήταν τοπική, δημιουργούσε δυσλειτουργία. Το πρόβλημα αυτό μπορούσε να λυθεί με μετατροπή του χαρακτηριστικού timestamp στους πίνακες Gastouri και Kavos της βάσης από τύπου timestamp σε timestampz (επιστρέφει μετά από αίτηση την ώρα από UTC σε τοπική) ή με εισαγωγή της ζώνης ώρας (time zone) μέσα στο ερώτημα SQL. Τελικά, επιλέχθηκε ο δεύτερος τρόπος, όπως φαίνεται παρακάτω:

“SELECT A.CONSUMPTION_KAVOS, A.TIMESTAMP AT TIME ZONE 'UTC' AT TIME ZONE 'Europe/Bucharest' FROM STATISTICS AS A WHERE TIMESTAMP>=\$1 AND TIMESTAMP<=\$2”

Κεφάλαιο 5

Συμπεράσματα- Μελλοντικές Τροποποιήσεις

Ολοκληρώνοντας την παρουσίαση της εφαρμογής Water Tanks, θα γίνει μια αναφορά στα αποτελέσματα της υλοποίησής της καθώς και στις μελλοντικές επεκτάσεις που θα μπορούσαν να γίνουν.

Η εφαρμογή Water Tank είναι μια προσπάθεια να οπτικοποιηθούν οι δεξαμενές ενός δικτύου ύδρευσης ως αναγκαιότητα για τη σωστή λειτουργία του. Μέσω της εφαρμογής, πέρα από τη γνώση ανά πάσα στιγμή της ποσότητας του νερού που είναι διαθέσιμη σε κάθε δεξαμενή, λύνουμε και το πρόβλημα της ανίχνευσης βλαβών στις δεξαμενές. Η δυνατότητα αυτή δεν υπήρχε μέχρι στιγμής καθώς ο μόνος τρόπος να ενημερωθούν οι διαχειριστές του συστήματος για την ύπαρξη βλάβης ήταν μέσω των χρηστών του συστήματος. Επιπρόσθετα, προσφέρει στατιστικά για τις καταναλώσεις σε κάθε δεξαμενή, κάτι που μπορεί να βοηθήσει στην κατανόηση της χρήσης του δικτύου η οποία μεταβάλλεται ανά εποχή και περιοχή του νησιού ώστε να γίνουν οι κατάλληλες βελτιώσεις στις υποδομές και να προσφέρονται στους χρήστες καλύτερες υπηρεσίες. Ο σχεδιασμός και η υλοποίηση της εφαρμογής στηρίχθηκαν πάνω στην επιλογή του NodeJS ως εξυπηρετητή και βιβλιοθηκών της JavaScript, όπως η D3. Το αποτέλεσμα, τόσο ως προς την οπτική απεικόνιση αλλά και ως προς το λειτουργικό κομμάτι, είναι ικανοποιητικό και εκπληρώνει πλήρως τον σκοπό για τον οποίο αναπτύχθηκε η εφαρμογή. Στο σημείο αυτό, θα πρέπει να αναφερθεί ότι υπάρχουν και άλλες δυνατότητες οι οποίες δεν υλοποιήθηκαν είτε λόγω χρόνου είτε λόγω προβλημάτων που εμφανίστηκαν κατά την υλοποίησή τους. Μερικές από αυτές είναι δευτερευούσης

σημασίας και εύκολες στην υλοποίηση, άλλες είναι πιο σημαντικές αλλά και δυσκολότερες στην υλοποίηση. Η εφαρμογή θα μπορούσε να κάνει χρήση του Google Maps API, το οποίο επιτρέπει στην εφαρμογή να έχει πρόσβαση σε δεδομένα και λειτουργίες των Google Maps. Αντί λοιπόν για τη σταθερή εικόνα που χρησιμοποιείται και πάνω σε αυτή έχουν σχεδιαστεί οι δεξαμενές, θα μπορούσε να γίνει χρήση μιας πιο αυτοματοποιημένης διαδικασίας εισάγοντας στον χάρτη το ακριβές γεωγραφικό στίγμα της κάθε δεξαμενής. Έτσι, θα μπορούσε η εφαρμογή να έχει περισσότερα στατιστικά όπως η μέση κατανάλωση σε κάθε δεξαμενή ανά μήνα. Η υλοποίησή της θα μπορούσε να βασιστεί στον σχεδιασμό που έχει γίνει για τη δημιουργία των στατιστικών με τη μέση κατανάλωση ανά ημέρα και εβδομάδα. Επιπλέον, θα μπορούσε να έχει δημιουργηθεί κατάλληλο μήνυμα που θα εμφανίζεται στον χρήστη και θα τον ενημερώνει σε περίπτωση που η εγγραφή της κατανάλωσης ανά λεπτό στον πίνακα statistics της βάσης δεδομένων, δεν είναι δυνατή. Τέλος, η πιθανή εγκατάσταση ενός επιπλέον αισθητήρα που θα μετράει την ποσότητα του νερού που εξέρχεται από τη δεξαμενή, θα έδινε τη δυνατότητα δημιουργίας ενός συναγερμού που θα ενεργοποιείται σε περίπτωση διαρροής νερού από τη δεξαμενή. Στην παρούσα φάση, η δημιουργία ενός τέτοιου συναγερμού δεν είναι εφικτή.

Βιβλιογραφία

<https://www.w3schools.com/nodejs/>

<https://nodejs.org/en/>

<https://www.geeksforgeeks.org/nodejs/>

<https://www.geeksforgeeks.org/javascript/>

<https://www.w3schools.com/js/>

<https://codepen.io/b-r-y/pen/MWpjzRM>

<https://www.freecodecamp.org/news/what-is-npm-a-node-package-manager-tutorial-for-beginners/>

<https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js>

<https://d3js.org/>

<https://www.maxbotix.com/articles/how-ultrasonic-sensors-work.htm>

<https://www.instructables.com/Measuring-water-level-with-ultrasonic-sensor/>

<https://www.ablic.com/en/semicon/products/sensor/magnetism-sensor-ic/intro/>

<https://wiki.seeedstudio.com/Water-Flow-Sensor/>

<https://blog.logrocket.com/crud-rest-api-node-js-express-postgresql/>

<https://observablehq.com/@d3/bar-chart>

<https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-timestamp/>

<https://openai.com/>

Συντμήσεις

API	Application Programming Interface
CLI	Command Line Interface
CSV	Comma Separated Values
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
ID	Identity Document
JSON	JavaScript Object Notation
NPM	Node Package Manager
SQL	Structured Query Language
URL	Uniform Resource Locator

Γλωσσάρι Ξενικών Όρων

API	Διασύνδεση εφαρμογής
Application	Εφαρμογή
Array	Πίνακας
Backend	Τμήμα εφαρμογής που τρέχει στο παρασκήνιο
Bar chart	Ραβδόγραμμα
Client	Πελάτης
Cross origin	Διασταυρούμενης προέλευσης
Frameworks	Πλαίσια
Frontend	Διεπαφή της εφαρμογής
Google Maps	Χάρτες Google
Query	Ερώτημα
Mean consumption	Μέση κατανάλωση
Middleware	Ενδιάμεσο λογισμικό
Module	Ενότητα
Navigation bar	Γραμμή πλοήγησης
Operating system	Λειτουργικό σύστημα
Parser	Αναλυτής
Primary key	Πρωτεύον κλειδί
Response	Απάντηση
Status	Κατάσταση
Timestamp	Χρονοσφραγίδα

ΠΑΡΑΡΤΗΜΑ

App.js

```
var counter = 0;

const express = require("express");
const path = require("path");
const bodyParser = require("body-parser");
const fs = require("fs");
const app = express();
const publicDirectoryPath = path.join(__dirname, "../public");
app.use(express.static(publicDirectoryPath));
app.use(bodyParser.json({ type: "application/json" }));
app.use(
  bodyParser.urlencoded({
    extended: true,
  })
);
const cors = require("cors");
const corsOptions = {
  origin: "*",
  credentials: true,
  optionSuccessStatus: 200,
};
app.use(cors(corsOptions));
const Pool = require("pg").Pool;
const config = {
  host: "localhost",
  user: "postgres",
  database: "deyak_tanks",
  port: 5432,
};

config.password = "1234";
const pool = new Pool(config);
```

```

app.get("/currentPreviousMeasurementsTank1", async (req, res) => {
  try {
    const template =
      "SELECT A.WATER_VOLUME,A.TIMESTAMP AT TIME ZONE 'UTC' AT TIME ZONE
'Europe/Bucharest',A.WATER_RATE FROM GASTOURI AS A WHERE A.TIMESTAMP<=$1 ";
    const response = await pool.query(template, [req.query.q]);
    if (response.rowCount == 0) {
      res.json({
        status: "notfound",
        searchTerm: req.query.q,
      });
    } else {
      res.json({
        status: "ok",
        results: response.rows.slice(
          response.rows.length - 2,
          response.rows.length
        ),
      });
    }
  } catch (err) {
    console.error("Error running query" + err);
  }
});

```

```

app.get("/currentPreviousMeasurementsTank2", async (req, res) => {
  try {
    const template =
      "SELECT A.WATER_VOLUME,A.TIMESTAMP AT TIME ZONE 'UTC' AT TIME ZONE
'Europe/Bucharest',A.WATER_RATE FROM KAVOS AS A WHERE A.TIMESTAMP<=$1";
    const response = await pool.query(template, [req.query.q]);
    if (response.rowCount == 0) {
      res.json({

```

```

        status: "notfound",
        searchTerm: req.query.q,
    });
} else {
    res.json({
        status: "ok",
        results: response.rows.slice(
            response.rows.length - 2,
            response.rows.length
        ),
    });
}
} catch (err) {
    console.error("Error running query" + err);
}
});

```

```

app.get("/consumptionRegistration", async (req, res) => {
    try {
        const template =
            "INSERT INTO STATISTICS (TIMESTAMP, CONSUMPTION_GASTOURI,
CONSUMPTION_KAVOS) VALUES ($1,$2,$3)";
        const response = await pool.query(template, [
            req.query.q1,
            req.query.q2,
            req.query.q3,
        ]);
        if (response.rowCount == 0) {
            res.json({
                status: "notfound",
                searchTerm: req.query.q,
            });
        } else {
            res.json({

```

```

        status: "ok",
    });
}
} catch (err) {
    console.error("Error running query" + err);
}
});

app.get("/statisticsHourlyTank1", async (req, res) => {
    try {
        const template =
            "SELECT A.CONSUMPTION_GASTOURI,A.TIMESTAMP AT TIME ZONE 'UTC' AT TIME
            ZONE 'Europe/Bucharest' FROM STATISTICS AS A WHERE TIMESTAMP>=$1 AND
            TIMESTAMP<=$2";

        const response = await pool.query(template, [req.query.q1, req.query.q2]);
        if (response.rowCount == 0) {
            res.json({
                status: "notfound",
                searchTerm: req.query.q,
            });
        } else {
            var sum;

            var values = [];
            sum = 0;
            for (var i = 0; i < response.rows.length; i++) {
                sum = sum + response.rows[i].consumption_gastouri;

                values[i] = response.rows[i].consumption_gastouri;
            }

            var meanConsumption = sum / response.rows.length;
            var timestamp = response.rows[0].timezone;

```

```

    counter++;
    res.json({
      status: "ok",
      results: {
        timestamp: timestamp,
        consumption: meanConsumption,
      },
    });
  }

} catch (err) {
  console.error("Error running query" + err);
}
});

app.get("/statisticsDailyTank1", async (req, res) => {
  try {
    const template =
      "SELECT A.CONSUMPTION_GASTOURI,A.TIMESTAMP AT TIME ZONE 'UTC' AT TIME
ZONE 'Europe/Bucharest' FROM STATISTICS AS A WHERE TIMESTAMP>=$1 AND
TIMESTAMP<=$2";

    const response = await pool.query(template, [req.query.q1, req.query.q2]);
    if (response.rowCount == 0) {
      res.json({
        status: "notfound",
        searchTerm: req.query.q,
      });
    } else {
      var sum;

      var values = [];
      sum = 0;
      for (var i = 0; i < response.rows.length; i++) {
        sum = sum + response.rows[i].consumption_gastouri;

```

```

        values[i] = response.rows[i].consumption_gastouri;
    }

    var meanConsumption = sum / response.rows.length;

    var timestamp = response.rows[0].timezone;

    counter++;

    res.json({
        status: "ok",
        results: {
            timestamp: timestamp,
            consumption: meanConsumption,
        },
    });
}

} catch (err) {
    console.error("Error running query" + err);
}

});

app.get("/statisticsHourlyTank2", async (req, res) => {
    try {
        const template =
            "SELECT A.CONSUMPTION_KAVOS,A.TIMESTAMP AT TIME ZONE 'UTC' AT TIME ZONE
'Europe/Bucharest' FROM STATISTICS AS A WHERE TIMESTAMP>=$1 AND TIMESTAMP<=$2";
        const response = await pool.query(template, [req.query.q1, req.query.q2]);
        if (response.rowCount == 0) {
            res.json({
                status: "notfound",
            });
        }
    } catch (err) {
        console.error("Error running query" + err);
    }
});

```

```

        searchTerm: req.query.q,
    });
} else {
    var sum;

    var values = [];
    sum = 0;
    for (var i = 0; i < response.rows.length; i++) {

        sum = sum + response.rows[i].consumption_kavos;
        values[i] = response.rows[i].consumption_kavos;
    }

    var meanConsumption = sum / response.rows.length;
    var timestamp = response.rows[0].timezone;

    res.json({
        status: "ok",
        results: {
            timestamp: timestamp,

            consumption: meanConsumption,
        },
    });
}
} catch (err) {
    console.error("Error running query" + err);
}
});

app.get("/statisticsDailyTank2", async (req, res) => {
    try {
        const template =

```

```

"SELECT A.CONSUMPTION_KAVOS,A.TIMESTAMP AT TIME ZONE 'UTC' AT TIME ZONE
'Europe/Bucharest' FROM STATISTICS AS A WHERE TIMESTAMP>=$1 AND TIMESTAMP<=$2";
const response = await pool.query(template, [req.query.q1, req.query.q2]);
if (response.rowCount == 0) {
  res.json({
    status: "notfound",
    searchTerm: req.query.q,
  });
} else {
  var sum;

  var values = [];
  sum = 0;
  for (var i = 0; i < response.rows.length; i++) {

    sum = sum + response.rows[i].consumption_kavos;
    values[i] = response.rows[i].consumption_kavos;
  }

  var meanConsumption = sum / response.rows.length;
  var timestamp = response.rows[0].timezone;
  res.json({
    status: "ok",
    results: {
      timestamp: timestamp,

      consumption: meanConsumption,
    },
  });
}
} catch (err) {
  console.error("Error running query" + err);
}
});

```

```

app.listen(3000, () => {
  console.log("Deyak tanks application starting..");
  console.log("Nodejs server is up on port 3000");
});

app.listen(app.get("port"), () => {
  console.log("Connection established in port" + app.get("port"));
});

```

Index.html

```

<!doctype html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Tank Πρωταρχή Εργασία</title>
  <link rel="icon" href="/assets/deyak_logo.png" type="image/x-icon">
  <link rel="stylesheet" href="/styles/style.css">
  <script src="/libs/jquery-3.5.1.js"></script>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-
gH2yIJqKdNHPEq0n4Mqa/HGKIhSkIHeL5AyhkYV8i59U5AR6csBvApHHNI/vI1Bx"
crossorigin="anonymous">
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-
A3rJD856KowSb7dwlZdYEkO39Gagi7vIsF0jrRAoQmDCKtQBHUuLZ9AsSv4jD4Xa"
    crossorigin="anonymous"></script>
  <script src="https://cdn.jsdelivr.net/npm/d3@7"></script>
</head>

```

```

<body>
<div>
<nav class="navbar navbar-expand-sm navbar-custom">
<div class="container-fluid">
<a class="navbar-brand" href="#">
    WATER-TANKS
</a>

<div class="collapse navbar-collapse" id="navbarSupportedContent">
<ul class="navbar-nav me-auto mb-2 mb-lg-0">
<li class="nav-item dropdown">
<a class="nav-link dropdown-toggle" id="TanksMenu" href="#" data-toggle="tooltip"
title="Water tanks list"
id="navbarDropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false">
    Tanks
</a>
<ul class="dropdown-menu" aria-labelledby="navbarDropdown">
<li><a class="dropdown-item" id='gastouri' href="#" onclick=clicktank(1)>Gastouri</a></li>
<li><a class="dropdown-item" id='kavos' href="#" onclick=clicktank(2)>Kavos</a></li>
<li><a class="dropdown-item" id='sokraki' href="#" onclick="">Sokraki</a></li>
<li><a class="dropdown-item" id='ermones' href="#" onclick="">Ermones</a></li>
</ul>
</li>
<li class="nav-item dropdown">
<a class="nav-link dropdown-toggle" href="#" data-toggle="tooltip" title="Alarm and Language
options"
id="options" role="button" data-bs-toggle="dropdown" aria-expanded="false">
    Options
</a>
<ul class="dropdown-menu" aria-labelledby="navbarDropdown">
<li><a class="dropdown-item" id="alarms" href="#" onclick=setAlarmLevels()>Set Alarm
Levels</a></li>
<li><a class="dropdown-item" id="language" href="#" onclick=changeLanguage()>Set
Language Greek</a></li>

```

```

        </ul>
    </li>
    <li class="nav-item">
        <a class="nav-link" id="about" onclick=aboutForm() href="#">About</a>
    </li>
</ul>
</div>
</div>
</nav>
</div>

```

```

<div class="paper">
    <div class="button-form1">
        <input type="button" class="tank1" id="tank1" name="tank1-button" value=" onclick=clicktank(1)
            style="background-color:white; opacity:1; border-radius: 50%; width:20px; height:20px;">
    </div>
    <div class="button-form2">
        <input type="button" class="tank2" id="tank2" name="tank2-button" value=" onclick=clicktank(2)
            style="background-color:white; opacity:1; border-radius: 50%; width:20px; height:20px;">
    </div>
    <div class="button-form3">
        <input type="button" class="tank3" id="tank3" name="tank3-button" value="
            style="background-color:white; opacity:1; border-radius: 50%; width:20px; height:20px;">
    </div>
    <div class="button-form4">
        <input type="button" class="tank4" id="tank4" name="tank4-button" value="
            style="background-color:white; opacity:1; border-radius: 50%; width:20px; height:20px;">
    </div>
    <div class="setAlarm-form">
        <input type="text" class="alarm-label" id="alarmLevels" readonly="readonly" tabindex="-1"
value="Alarm Levels"
            style="text-align: center; pointer-events:none; border-style: hidden; background-color:rgba(38, 58,
70,0.7); color:white; margin-left: 10px; width: 360px; margin-top: 15px;">
        <button class="alarm-close-icon" id="close-form"

```

```

        style="background-color:rgba(0, 0, 0,0.6); color: white; margin-top: 15px; ">
        &#10006;
    </button>
    <hr style=" width:410px; height:3px; background-color: white;">
    <input type="text" class="overflow-label" id="alarmMaxLevel" readonly="readonly" tabindex="-1"
value="Max Level"
        style="border:1 px; text-align: center; pointer-events:none; background-color: black ; color:white;
margin-left: 10px; width: 92px; margin-top: 10px;">
    <input type="text" class="overflow-value"
        style="border:1 px; text-align: center; background-color: black ; color:white; margin-left: 0px;
width: 92px; margin-right: 0px;">
    <input type="text" class="drainage-label" id="alarmMinLevel" tabindex="-1" readonly="readonly"
value="Min Level"
        style="border:1 px; text-align: center; pointer-events:none; background-color: black ; color:white;
margin-left: 10px; width: 92px; margin-top: 5px;">
    <input type="text" class="drainage-value"
        style="border:1 px; text-align: center; background-color: black ; color:white; margin-left: 0px;
width: 92px; margin-right: 0px;">
    </div>
    <div class="alarm-info-form" style="background-color:red;">
    <input type="text" class="alarm-info-label" value="" readonly="readonly"
        style="text-align: center; pointer-events:none; border-style: hidden; background-color: inherit;
color:white; margin-left: 10px; width: 350px; margin-top: 10px;">
    <div class="acknowledge-form" style="background-color:red;">
    <button class="acknowledge-button" style="color: white;">
        ACK
    </button>
    </div>
    </div>
    <div class="d3-form" id="d3-form-div">
    <input type="text" class="d3-label" readonly="readonly" tabindex="-1" value=""
        style="text-align: center; pointer-events:none; border-style: hidden; background-color:rgba(38, 58,
70,0.7); color:white; margin-left: 10px; width: 545px; margin-top: 15px;">
    <button class="d3-close-icon" id="close-form"

```

```

        style="background-color:rgba(0, 0, 0,0.6); color: white; margin-top: 15px; ">
        &#10006;
    </button>
    <hr style=" width:600px; height:3px; background-color: white;">
    <label class='x-axis'
        style='width:60px; text-align:right; position:fixed; top: 770px; left: 1789px; font-size: 13px;
background-color:rgb(38,58,70); opacity: 1; color: white;'></label>
</div>

<div class="tank-form">
    <input type="text" class="tank-label" tabindex="-1" readonly="readonly"
        style="text-align: center; pointer-events:none; border-style: hidden; background-color:rgba(38, 58,
70,0.7); color:white; margin-left: 10px; width: 320px; margin-top: 15px;">
    <button class="tank-close-icon" id="close-form"
        style="background-color:rgba(0, 0, 0,0.6); color: white; margin-top: 15px; ">
        &#10006;
    </button>
    <hr style=" width:370px; height:3px; background-color: white;">
    <div style="border-style: solid; background-color:rgba(85, 105, 106, 0.5); border-color: black; height:
auto; width: 352px; margin-top:5px; border-width:1px; margin-left: 10px; border-radius: 10px;">
        <input type="text" class="info-tank-maxVolume"tabindex="-1" id="tankMaxVolume" value="Max
Volume of Tank" readonly="readonly" style="border:1 px; text-align: center; pointer-events:none;
background-color: black ; color:white; margin-left: 10px; width: 165px; margin-top: 5px; margin-
left:10px; margin-bottom:5px; margin-right:0px;">
            <input type="text" class="value-maxVolume" tabindex="-1" readonly="readonly"
                style="border:1 px; text-align: center; pointer-events:none; background-color: black ; color:white;
margin-left: 0px; width: 165px; margin-right: 0px; margin-bottom:5px;">

    </div>
</div>
<div
    style="border-style: solid; background-color:rgba(85, 105, 106, 0.5); border-color: black; height:
auto; width: 352px; margin-top:305px; border-width:1px; margin-left: 10px; border-radius: 10px;">
    <input type="text" class="info-percentage" id="fillingLevel" tabindex="-1" value="Filling Level"
        readonly="readonly"

```

```

        style="border:1 px;text-align: center; pointer-events:none; background-color: black ; color:white;
margin-left: 10px; width: 165px; margin-top: 10px;">
        <input type="text" class="value-percentage" tabindex="-1" readonly="readonly"
        style="border:1 px;text-align: center; pointer-events:none; background-color: black ; color:white;
margin-left: 0px; width: 165px; margin-right: 0px;">
        <input type="text" class="info-volume" id="currentVolume" tabindex="-1" readonly="readonly"
        value="Current Volume"
        style="border:1 px;text-align: center; pointer-events:none; background-color: black ; color:white;
margin-left: 10px; width: 165px; margin-top: 5px;">
        <input type="text" class="value-volume" tabindex="-1" readonly="readonly"
        style="border:1 px;text-align: center; pointer-events:none; background-color: black ; color:white;
margin-left: 0px; width: 165px; margin-right: 0px;">
        <input type="text" class="info-rate" id="currentRate" tabindex="-1" readonly="readonly"
value="Current Rate"
        style="border:1 px;text-align: center; pointer-events:none; background-color: black ; color:white;
margin-left: 10px; width: 165px; margin-top: 5px;">
        <input type="text" class="value-rate" tabindex="-1" readonly="readonly"
        style="border:1 px;text-align: center; pointer-events:none; background-color: black ; color:white;
margin-left: 0px; width: 165px; margin-right: 0px;">
        <input type="text" class="info-consumption" id="consumptionPerMinute" tabindex="-1"
readonly="readonly"
        value="Consumption Per Minute"
        style="border:1 px;text-align: center; pointer-events:none; background-color: black ; color:white;
margin-left: 10px; width: 165px; margin-top: 5px;">
        <input type="text" class="value-consumption" tabindex="-1" readonly="readonly"
        style="border:1 px;text-align: center; pointer-events:none; background-color: black ; color:white;
margin-left: 0px; width: 165px; margin-bottom: 10px; margin-right: 0px;">
    </div>
    <div id="tank" class="tank">
    </div>
    <div class="d3-button1-form">
        <input type="button" class="d3Button1" id="dailyConsumptionGraph" tabindex="-1" value="Daily
Consumption Graph"

```

```

        style="z-index:3000; border:1 px; border-color:white; text-align: center; background-color: black ;
color:white; margin-left: 10px; width: 160px; margin-top: 10px; height: 20px;">
    </div>
    <div class="d3-button1-date">
        <input type="date" class="d3date1" id="d3date1" name="d3-date1"
            style="border:1 px; border-color:white; background-color: black ; color:white; text-align: center;
margin-left: 10px; width: 160px; margin-top: 10px; height: 20px;">
    </div>
    <div class="submit">
        <button type="button" id="submit" onclick=submitDate()
            style="border:1 px; border-color:white; background-color: black ; color:white; text-align: center;
margin-left: 10px; width: 160px; margin: 10px; height: 20px;">Submit</button>
    </div>
    <div class="d3-button2-form">
        <input type="button" class="d3Button2" id="weeklyCosumptionGraph" tabindex="-1"
value="Weekly Consumption Graph"
            readonly="readonly"
            style="border:1 px; border-color:white; text-align: center; background-color: black ; color:white;
margin-left: 10px; width: 160px; margin-top: 10px; height: 20px;">
    </div>
</div>
<div class="status-bar">
    <label class='info-1' id="currentTimestamp" style="width:200px; text-align:right; color:
white;">Current Timestamp
        :</label>
    <label class='info-2' style="width:500px; text-align:left;"></label>
    <label class='info-3' id="maxLevel" style="width:125px; text-align:right; color: white;">Max Level
:</label>
    <label class='info-4' style="width:40px; text-align:left;"></label>
    <label class='info-5' id="minLevel" style="width:140px; text-align:right; color: white;">Min Level
:</label>
    <label class='info-6' style="width:100px; text-align:left;"></label>
</div>
<div id="map" class="map">

```

```

    
  </div>
</div>
<div class="toolt">
  <div class="tooltipText"></div>
  <div class="imageContainer">
    <img class="imageTooltip" src="" alt="tank">
  </div>
</div>
</div>
</div>
<div class="tooltipDateDiv">
  <h2 class="tooltipDate" , id="toolTipDate">
</h2>
</div>
</div>
<div class="about" onclick=aboutForm()>
  <br>
  <br>
  <br>
  
  <br>
  <br>
  <br>
  <h1> ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ</h1>
  <br>
  <h2> Δημιουργία λογισμικού παρακολούθησης, απεικόνισης και αποστολής ειδοποιήσεων για
δεξαμενές δικτύου ύδρευσης
</h2>
  <br>
  <br>
  <p> ΜΥΡΩΝΗΣ ΙΩΑΝΝΗΣ
  <br>
  <br>
  Π2018221

```

```

    <br>
    <br>
    <br>
    Επιβλέπων : Αυλωνίτης Μάρκος
  </p>
</div>
</body>

<script src='./source/time.js'></script>
<script src='./source/functions.js'></script>
<script src='./source/drawTank.js'></script>
<script src='./source/main.js'></script>
<script src='./source/tooltip.js'></script>
<script src='./source/d3form.js'></script>
<script src='./source/language.js'></script>
<script src='./source/about.js'></script>
<script>
  const div = d3.selectAll("div");
</script>

</html>

```

Main.js

```
$(window).on("load", calculationRegistrationOfConsumption());
```

```

calculateScreenHeight = () => {
  var calcScreenHeight = window.innerHeight;
  $(".paper").css("height", calcScreenHeight - 70);

  $(".image").css("height", calcScreenHeight - 70);
  $(".image").css("width", (calcScreenHeight - 70) * 0.9);
  $(".status-bar").css("top", calcScreenHeight - 20);
}

```

```

$(".button-form1").css("left", (410 * window.innerHeight) / 980);
$(".button-form1").css("top", (560 * window.innerHeight) / 980);
$(".button-form2").css("left", (690 * window.innerHeight) / 980);
$(".button-form2").css("top", (850 * window.innerHeight) / 980);
};

```

```

calculateScreenHeight();

```

```

$(window).on("load", calculateScreenHeight());

```

```

var d3Period = "none";

```

```

clicktank = (tankname) => {
  cleartanks();
  var tankNameClass;
  tankNameClass = ".tank" + tankname;///create class name

```

```

$(".tank-form").css("display", "none");
$(".alarm-info-label").val("");
$(".alarm-info-form").css("animation", "glowing none");
$(".d3date1").attr("disabled", false);
$(".d3Button1").attr("disabled", false);
$(".d3Button2").attr("disabled", false);
$(".alarm-info-form").css("display", "none");
$(".alarm-info-label").css("display", "none");
$(".tooltipDateDiv").css("display", "none");
$(".submit").css("display", "none");
$(".d3-button1-date").css("display", "none");
if (tankname != activeTank) {
  $(".d3-form").css("display", "none");
}

```

```

$(".tank-form").css("display", "block");

```

```

$(tankNameClass).css("width", "30px");
$(tankNameClass).css("height", "30px");
activeTank = tankname;
drawTank(tankPercentage[tankname - 1]);

if (language == "english") {
    $(".tank-label").val(tanknames[tankname - 1]);
} else if (language == "greek") {
    $(".tank-label").val(tanknamesgr[tankname - 1]);
}
};

$(".tank-close-icon").click(function () {
    cleartanks();
    $(".alarm-info-label").val("");
    $(".alarm-info-form").css("animation", "glowing none");
    $(".tooltipDateDiv").css("display", "none");
    $(".d3date1").prop("value", "");
    $(".submit").css("display", "none");
    $(".d3-button1-date").css("display", "none");
    $(".alarm-info-form").css("display", "none");
    $(".alarm-info-label").css("display", "none");
    activeTank = 0;
    $(".d3-form").css("display", "none");
});

$(".d3-close-icon").click(function () {
    $(".d3-form").css("display", "none");
    $(".submit").css("display", "none");

    $(".d3Button1").prop("disabled", false);
    $(".d3Button2").prop("disabled", false);

    $(".d3date1").attr("disabled", false);

```

```

$(".d3-button1-date").css("display", "none");

var previousSvg = document.getElementById("graph"); // check for a previous svg existence
if (previousSvg != null) {
    previousSvg.remove();
}
});

$(".d3Button1").click(function () {
    consumptionArray = [];

    var previousSvg = document.getElementById("graph"); // check for a previous svg existence
    if (previousSvg != null) {
        previousSvg.remove();
    }
    d3Period = "daily";

    $(".d3Button1").prop("disabled", true);
    $(".d3Button2").prop("disabled", true);
    $(".d3-form").css("display", "none");
    $(".d3date1").attr("disabled", false);
    $(".d3date1").val("");

    $(".d3-button1-date").css("display", "block");
    $(".tooltipDateDiv").css("display", "block");

    if (language == "english") {
        $(".tooltipDate").html("Please select day");
    } else if (language == "greek") {
        $(".tooltipDate").html("Παρακαλώ επιλέξτε ημέρα");
    }
});

$(".d3date1").change(function () {

```

```

$(".submit").css("display", "block");
});

function daily() {
    $(".d3date1").attr("disabled", true);
    $(".tooltipDateDiv").css("display", "none");
    $(".d3-form").css("display", "block");
    $(".submit").css("display", "none");
    d3Period = "none";

    d3graphDaily("daily");

    if (language == "english") {
        $(".d3-label").val(
            "Daily Consumption of " + tanknames[activeTank - 1] + " Tank"
        );
    } else if (language == "greek") {
        $(".d3-label").val(
            "Ημερήσια Κατανάλωση Δεξαμενής " + tanknamesgr[activeTank - 1]
        );
    }
}

$(".d3Button2").click(function () {
    consumptionArray = [];

    var previousSvg = document.getElementById("graph"); // check for a previous svg existence
    if (previousSvg != null) {
        previousSvg.remove();
    }
    d3Period = "weekly";

    $(".d3Button1").prop("disabled", true);
    $(".d3Button2").prop("disabled", true);

```

```

$(".d3-form").css("display", "none");
$(".d3date1").attr("disabled", false);
$(".d3date1").val("");

$(".d3-button1-date").css("display", "block");
$(".tooltipDateDiv").css("display", "block");
if (language == "english") {
    $(".tooltipDate").html("Please select the date the week starts");
} else if (language == "greek") {
    $(".tooltipDate").html("Παρακαλώ επιλέξτε την αρχή της εβδομάδας");
}
});

function submitDate() {
    if (d3Period == "daily") {
        daily();
    } else if (d3Period == "weekly") {
        weekly();
    }
}

function weekly() {
    $(".d3date1").attr("disabled", true);
    $(".tooltipDateDiv").css("display", "none");
    $(".d3-form").css("display", "block");
    $(".submit").css("display", "none");

    d3graphWeekly("weekly");

    if (language == "english") {
        $(".d3-label").val(
            "Weekly Consumption of " + tanknames[activeTank - 1] + " Tank"
        );
    } else if (language == "greek") {

```

```

$(".d3-label").val(
    "Εβδομαδιαία Κατανάλωση Δεξαμενής " + tanknamesgr[activeTank - 1]
);
}

d3Period = "none";
}
// clear tank form and reset buttons to 20px
cleartanks = () => {
    $(".tank-form").css("display", "none");
    $(".tank1").css("width", "20px");
    $(".tank1").css("height", "20px");
    $(".tank2").css("width", "20px");
    $(".tank2").css("height", "20px");
    $(".tank3").css("width", "20px");
    $(".tank3").css("height", "20px");
    $(".tank4").css("width", "20px");
    $(".tank4").css("height", "20px");
};

$(".alarm-close-icon").click(function () {
    var minValue = $(".drainage-value").css("background-color");
    var maxValue = $(".overflow-value").css("background-color");

    if (minValue == "rgb(0, 0, 0)" && maxValue == "rgb(0, 0, 0)") {
        $(".setAlarm-form").css("display", "none");
    } else {
        alert("Please insert correct values for alarm levels");
    }
});

```

Functions.js

```
var soundInterval1;
var soundInterval2;
var acknowledgedTank = [];
var audio = new Audio("./assets/alarm.wav");
var maxVolumeTank = [300,200];
var activeTank = 0;
var actualTimestamp;
var initialTimestamp;
var tanknames = ["GASTOURI", "KAVOS", "SOKRAKI", "ERMONES"];
var tanknamesgr = ["ΓΑΣΤΟΥΡΙ", "ΚΑΒΟΣ", "ΣΩΚΡΑΚΙ", "ΕΡΜΟΝΕΣ"];
var numberOfTanks = 4;
var tankPercentage = [];
var previousVolume = [];
var currentVolume = [];
var currentRate = [];
var currentConsumption = [];
var minLevel, maxLevel;
var coord1x = (400 * window.innerHeight) / 980 + 20;
var coord1y = (385 * window.innerHeight) / 980 + 20;
var coord2x = (690 * window.innerHeight) / 980 - 260;
var coord2y = (260 * window.innerHeight) / 980 - 140;

var coordinates = [
  [coord1x, coord1y],
  [coord2x, coord2y],
  [270, 310],
  [255, 490],
];

acknowledgedTank[0] = false;
```

```

acknowledgedTank[1] = false;
minLevel = 10;
maxLevel = 90;
$(".overflow-value").val(maxLevel);
$(".drainage-value").val(minLevel);

calculationRegistrationOfConsumption = () => {
  initialTimestamp = getTimestamp();
  $(".info-2").text(initialTimestamp); //load timestamp to status bar
  updateStatusBarLevels(); //load min and max level to status bar

  getCurrentAndPreviousValues(initialTimestamp);
// setInterval run every 60seconds
  setInterval(function () {
    actualTimestamp = getTimestamp();
    $(".info-2").text(actualTimestamp);
    getCurrentAndPreviousValues(actualTimestamp);
  }, 60000);
};

updateStatusBarLevels = () => {
  $(".info-4").text(maxLevel + " %");
  $(".info-6").text(minLevel + " %");
};

getCurrentAndPreviousValues = (currentTimestamp) => {
  fetch(
    "http://localhost:3000/currentPreviousMeasurementsTank1?q=" +
    currentTimestamp
  ).then((response) => {
    response.json().then((data) => { //destrignify the response and create object data with the results
      tankPercentage[0] = (100 * data.results[1].water_volume) / maxVolumeTank[0];
      previousVolume[0] = data.results[0].water_volume;
      currentVolume[0] = data.results[1].water_volume;
    });
  });
}

```

```

currentRate[0] = data.results[1].water_rate;

if (activeTank == 1) {
    drawTank(tankPercentage[0]);
}

fetch(
    "http://localhost:3000/currentPreviousMeasurementsTank2?q=" +
    currentTimestamp
).then((response) => {
    response.json().then((data) => {
        tankPercentage[1] =
            (100 * data.results[1].water_volume) / maxVolumeTank[1];
        previousVolume[1] = data.results[0].water_volume;
        currentVolume[1] = data.results[1].water_volume;
        currentRate[1] = data.results[1].water_rate;

        checkLevels();

        if (activeTank == 2) {
            drawTank(tankPercentage[1]);
        }

        setTimeout(function () {
            var consumption;
            calculateConsumption(
                previousVolume[0],
                currentVolume[0],
                currentRate[0],
                previousVolume[1],
                currentVolume[1],
                currentRate[1]
            );
        },

```

```

    if (activeTank == 1) {
        $(".value-consumption").val(
            currentConsumption[0].toFixed(3) + " lt/min"
        );
    } else {
        $(".value-consumption").val(
            currentConsumption[1].toFixed(3) + " lt/min"
        );
    }
    var timestampValue = data.results[1].timezone; //use database timestamp in consumption
registration

    consumptionRegistration(
        timestampValue,
        currentConsumption[0],
        currentConsumption[1]
    );
    }, 1000);
});
});
});
});
};

calculateConsumption = (
    waterVolumePrevious1,
    waterVolumeCurrent1,
    waterRateCurrent1,
    waterVolumePrevious2,
    waterVolumeCurrent2,
    waterRateCurrent2
) => {
    var consumption;
    var waterRateCoverted1;

```

```

waterRateCovered1 = waterRateCurrent1 / 1000;//convert to cubic metres
var waterRateCovered2;
waterRateCovered2 = waterRateCurrent2 / 1000;
////////insert data to currentConsumption table and cut decimal digits to 6 //////////
currentConsumption[0] =
    parseFloat(waterVolumePrevious1.toFixed(6)) +
    parseFloat(waterRateCovered1.toFixed(6)) -
    parseFloat(waterVolumeCurrent1).toFixed(6);
currentConsumption[1] =
    parseFloat(waterVolumePrevious2.toFixed(6)) +
    parseFloat(waterRateCovered2.toFixed(6)) -
    parseFloat(waterVolumeCurrent2).toFixed(6);
};

consumptionRegistration = (timestampValue, consumption1, consumption2) => {
    fetch(
        "http://localhost:3000/consumptionRegistration?q1=" +
        timestampValue +
        "&q2=" +
        consumption1 +
        "&q3=" +
        consumption2
    ).then((response) => {
        response.json().then((data1) => {});
    });
};

setAlarmLevels = () => {
    $(".setAlarm-form").css("display", "block");
    $(".overflow-value").val(maxLevel);
    $(".drainage-value").val(minLevel);
};

$(".overflow-value").change(function () {

```

```

tex = $(".overflow-value").val();

if (isNaN(tex) == true) {
    $(".overflow-value").css("background-color", "red");
} else if (parseInt(tex) > 99 || parseInt(tex) < 1) {
    $(".overflow-value").css("background-color", "red");
} else if (tex <= minLevel) {
    $(".overflow-value").css("background-color", "red");
} else {
    $(".overflow-value").css("background-color", "black");
    maxLevel = tex;
    updateStatusBarLevels();
    checkLevels();
}
});

```

```

$(".drainage-value").change(function () {
    tex = $(".drainage-value").val();

    if (isNaN(tex) == true) { // check if tex is number
        $(".drainage-value").css("background-color", "red");
    } else if (parseInt(tex) > 99 || parseInt(tex) < 1) {
        $(".drainage-value").css("background-color", "red");
    } else if (tex >= maxLevel) {
        $(".drainage-value").css("background-color", "red");
    } else {
        $(".drainage-value").css("background-color", "black");
        minLevel = tex;
        updateStatusBarLevels();
        checkLevels();
    }
});

```

```

checkLevels = () => {

```

```

if (activeTank != 0) { //reset all parameters if there is an active tank
    $(".tank-form").css("display", "none");
    $(".alarm-info-label").val("");
    $(".alarm-info-form").css("animation", "glowing none");

    $(".alarm-info-form").css("display", "none");
    $(".alarm-info-label").css("display", "none");

    clicktank(activeTank); // call clicktank in order to draw the active tank again
}
////////// if alarm enable for first time in tank gastouri enable sound alarm and glowing//////////
if (tankPercentage[0] < minLevel || tankPercentage[0] > maxLevel) {
    if (soundInterval1 == undefined && acknowledgedTank[0] == false) {
        soundInterval1 = setInterval(function () {///create sound alarm
            audio.play();
        }, 1);
    }
    //if alarm already exist in tank Gastouri disable glowing///
    if (acknowledgedTank[0] == true) {
        $(".alarm-info-form").css("animation", "glowing none");
    }

    $(".tank1").css("animation", "glowing 1500ms infinite");
}
else {
    $(".tank1").css("animation", "glowing none");

    acknowledgedTank[0] = false;
    /// check if soundinterval is enable///
    if (soundInterval1 != undefined) {
        clearInterval(soundInterval1);
        soundInterval1 = undefined;
    }
}
}

```

```

////////// if alarm enable for firts time in tank kavos enable sound alarm and glowing//////////
if (tankPercentage[1] < minLevel || tankPercentage[1] > maxLevel) {
  if (soundInterval2 == undefined && acknowledgedTank[1] == false) {
    soundInterval2 = setInterval(function () {///create sound alarm
      audio.play();
    }, 1);
  }
  //if alarm already exist in tank Gastouri disable glowing///
  if (acknowledgedTank[1] == true) {
    $(".alarm-info-form").css("animation", "glowing none");
  }

  $(".tank2").css("animation", "glowing 1500ms infinite");
} else {
  $(".tank2").css("animation", "glowing none");
  acknowledgedTank[1] = false;
  if (soundInterval2 != undefined) {
    clearInterval(soundInterval2);
    soundInterval2 = undefined;
  }
}
};

$(".acknowledge-button").click(function () {
  if (acknowledgedTank[activeTank - 1] == false) {
    if ((activeTank = 1) && soundInterval1 != undefined) {
      clearInterval(soundInterval1);

      $(".alarm-info-form").css("animation", "glowing none");
      $(".acknowledge-form").css("animation", "glowing none");
      soundInterval1 = undefined;
    } else if ((activeTank = 2) && soundInterval2 != undefined) {
      clearInterval(soundInterval2);

```

```

$(".alarm-info-form").css("animation", "glowing none");
$(".acknowledge-form").css("animation", "glowing none");
soundInterval2 = undefined;
}
acknowledgedTank[activeTank - 1] = true;
}
});

```

Drawtank.js

```

var canvas = document.createElement("canvas");

var ctx = canvas.getContext("2d");

canvas.height = 310;
canvas.width = 200;
document.getElementById("tank").appendChild(canvas);
document.getElementById("tank").style = "opacity:1";
drawTank = (y) => {
  if (activeTank == 1) {
    $(".value-percentage").val(tankPercentage[0].toFixed(3) + " %");
    $(".value-volume").val(currentVolume[0].toFixed(3) + " m^3");
    $(".value-rate").val(currentRate[0].toFixed(1) + " lt/min");
    $(".value-maxVolume").val(maxVolumeTank[0] + " m^3");
  } else {
    $(".value-percentage").val(tankPercentage[1].toFixed(3) + " %");
    $(".value-volume").val(currentVolume[1].toFixed(3) + " m^3");
    $(".value-rate").val(currentRate[1].toFixed(1) + " lt/min");
    $(".value-maxVolume").val(maxVolumeTank[1] + " m^3");
  }

  if (activeTank == 1) {
    $(".value-consumption").val(currentConsumption[0].toFixed(3) + " lt/min");
  }
}

```

```

    } else {
        $(".value-consumption").val(currentConsumption[1].toFixed(3) + " lt/min");
    }
    checkAlarmForm(y);

    var a = 0.3;

    var size = 100;
    var height = 200;
    var center = 100;
    var bottom = 290;

    function shadeColor(color, percent) {
        color = color.substr(1);
        var num = parseInt(color, 16),
            amt = Math.round(2.55 * percent),
            R = (num >> 16) + amt,
            G = ((num >> 8) & 0x00ff) + amt,
            B = (num & 0x0000ff) + amt;
        return (
            "#" +
            (
                0x1000000 +
                (R < 255 ? (R < 1 ? 0 : R) : 255) * 0x10000 +
                (G < 255 ? (G < 1 ? 0 : G) : 255) * 0x100 +
                (B < 255 ? (B < 1 ? 0 : B) : 255)
            )
            .toString(16)
            .slice(1)
        );
    }

    function draw() {
        ctx.clearRect(0, 0, canvas.width, canvas.height);

```

```
var wobble = (Math.sin(Date.now() / 250) * height) / 80;
```

```
drawCube(           //draw tank
  center,
  bottom,
  size,
  size,
  height,
  shadeColor("#000000", 20),
  false
);
if ((height * y) / 100 + wobble > maxVolumeTank - 5) {
  wobble = 0;
}
```

```
if (y > maxLevel || y < minLevel) {
  drawCube(           //draw water red
    center,
    bottom,
    size,
    size,
    (height * y) / 100 + wobble,
    "#ff0000",
    true
  );
} else {
  drawCube(           //draws water blue
    center,
    bottom,
    size,
    size,
    (height * y) / 100 + wobble,
    "#24b6ff",
    true
  );
}
```

```

    );
    console;
}

requestAnimationFrame(draw);
}
draw();

function hexToRgba(hex) {
    var c;

    if (/^#([A-Fa-f0-9]{3}){1,2}$/.test(hex)) {
        c = hex.substring(1).split("");
        if (c.length == 3) {
            c = [c[0], c[0], c[1], c[1], c[2], c[2]];
        }
        c = "0x" + c.join("");
        return (
            "rgba(" + [(c >> 16) & 255, (c >> 8) & 255, c & 255].join(",") + ",.5)"
        );
    }
    throw new Error("Bad Hex");
}

function drawCube(x, y, wx, wy, h, color, transparent) {
    ctx.beginPath();
    ctx.moveTo(x, y);
    ctx.lineTo(x - wx, y - wx * a);
    ctx.lineTo(x - wx, y - h - wx * a);
    ctx.lineTo(x, y - h * 1);
    ctx.closePath();
    ctx.fillStyle = shadeColor(color, -10);
    ctx.strokeStyle = color;
    ctx.stroke();
}

```

```

ctx.fill();

ctx.beginPath();
ctx.moveTo(x, y);
ctx.lineTo(x + wy, y - wy * a);
ctx.lineTo(x + wy, y - h - wy * a);
ctx.lineTo(x, y - h * 1);
ctx.closePath();
ctx.fillStyle = shadeColor(color, 10);
ctx.strokeStyle = shadeColor(color, 50);
ctx.stroke();
ctx.fill();

ctx.beginPath();
ctx.moveTo(x, y - h);
ctx.lineTo(x - wx, y - h - wx * a);
ctx.lineTo(x - wx + wy, y - h - (wx * a + wy * a));
ctx.lineTo(x + wy, y - h - wy * a);
ctx.closePath();
if (transparent) {
    ctx.fillStyle = hexToRGBA(color);
} else {
    ctx.fillStyle = shadeColor(color, 20);
}
ctx.strokeStyle = shadeColor(color, 60);
ctx.stroke();
ctx.fill();
}
};

checkAlarmForm = (y) => {
    $(".alarm-info-label").val("");
    var tankFormStatus = $(".tank-form").css("display");
    if (((y > maxLevel) || (y < minLevel)) && (tankFormStatus == "block")) {

```

```

var textMessage;
if (y > maxLevel) {
  if (language == "english") {
    textMessage = "Overflow!";
  } else if (language == "greek") {
    textMessage = "Υπερχείλιση!";
  }
} else {
  if (language == "english") {
    textMessage = "Drainage!";
  } else if (language == "greek") {
    textMessage = "Ελλειψη Νερού!";
  }
}

if (acknowledgedTank[activeTank - 1] == false) {
  $(".alarm-info-form").css("animation", "glowing 1500ms infinite");
}

$(".alarm-info-label").css("display", "block");

$(".alarm-info-form").css("display", "block");

if (language == "english") {
  $(".alarm-info-label").val("Alarm! Tank " + textMessage);
} else if (language == "greek") {
  $(".alarm-info-label").val("Συναγερμός! " + textMessage);
}
};

```

D3form.js

```
var m31 = [1, 3, 5, 7, 8, 10, 12];
```

```
var consumptionArray = [];
```

```
function BarChart(
  data,
  {
    x = (d, i) => i, // given d in data, returns the (ordinal) x-value
    y = (d) => d, // given d in data, returns the (quantitative) y-value
    title, // given d in data, returns the title text
    marginTop = 20, // the top margin, in pixels
    marginRight = 0, // the right margin, in pixels
    marginBottom = 30, // the bottom margin, in pixels
    marginLeft = 40, // the left margin, in pixels
    width = 640, // the outer width of the chart, in pixels
    height = 400, // the outer height of the chart, in pixels
    xDomain, // an array of (ordinal) x-values
    xRange = [marginLeft, width - marginRight], // [left, right]
    yType = d3.scaleLinear, // y-scale type
    yDomain, // [ymin, ymax]
    yRange = [height - marginBottom, marginTop], // [bottom, top]
    xPadding = 0.1, // amount of x-range to reserve to separate bars
    yFormat, // a format specifier string for the y-axis
    yLabel, // a label for the y-axis
    color = "currentColor", // bar fill color
  } = {}
) {
  // Compute values.
  const X = d3.map(data, x);
  const Y = d3.map(data, y);
```

```

// Compute default domains, and unique the x-domain.
if (xDomain === undefined) xDomain = X;
if (yDomain === undefined) yDomain = [0, d3.max(Y)];
xDomain = new d3.InternSet(xDomain);

// Omit any data not present in the x-domain.
const I = d3.range(X.length).filter((i) => xDomain.has(X[i]));

// Construct scales, axes, and formats.
const xScale = d3.scaleBand(xDomain, xRange).padding(xPadding);
const yScale = yType(yDomain, yRange);
const xAxis = d3.axisBottom(xScale).tickSizeOuter(0);
const yAxis = d3.axisLeft(yScale).ticks(height / 40, yFormat);

// Compute titles.
if (title === undefined) {
  const formatValue = yScale.tickFormat(100, yFormat);
  title = (i) => `${X[i]}\n${formatValue(Y[i])}`;
} else {
  const O = d3.map(data, (d) => d);
  const T = title;
  title = (i) => T(O[i], i, data);
}

var previousSvg = document.getElementById("graph"); // check for a previous svg existence
if (previousSvg !== null) {
  previousSvg.remove();
}
// create svg and import attributes
const svg = d3
  .select("#d3-form-div")
  .append("svg")
  .attr("id", "graph")
  .attr("width", width)

```

```

.attr("height", height)
.attr("viewBox", [0, 0, width, height])
.attr("style", "max-width: 100%; height: auto; height: intrinsic;");

svg
  .append("g")
  .attr("transform", `translate(${marginLeft},0)`)
  .call(yAxis)
  .call((g) => g.select(".domain").remove())
  .call((g) =>
    g
      .selectAll(".tick line")
      .clone()
      .attr("x2", width - marginLeft - marginRight)
      .attr("stroke-opacity", 0.1)
  )
  .call((g) =>
    g
      .append("text")
      .attr("x", -marginLeft)
      .attr("y", 10)
      .attr("fill", "currentColor")
      .attr("text-anchor", "start")
      .attr("font-size", 13)
      .text(yLabel)
  );
// create svg bars
const bar = svg
  .append("g")
  .attr("fill", color)
  .selectAll("rect")
  .data(I)
  .join("rect")
  .attr("x", (i) => xScale(X[i]))

```

```

.attr("y", (i) => yScale(Y[i]))
.attr("height", (i) => yScale(0) - yScale(Y[i]))
.attr("width", xScale.bandwidth());

if (title) bar.append("title").text(title);

svg
  .append("g")
  .attr("transform", `translate(0,${height - marginBottom})`)
  .call(xAxis);

return svg.node();
}

d3graphDaily = (graph) => {
  changexAxis(graph);

  consumptionArray = [];

  var i = 0;

  var selectedDate = $(".d3date1").prop("value"); // store the date from user selection
  var year = selectedDate.slice(0, 4);
  var month = selectedDate.slice(5, 7);
  var day = selectedDate.slice(8, 10);
  var hour = "00";
  var minute = "00";

  var timestampStart;
  var timestampEnd;
  var f = 0;

  var endHour;
  var hourInt;

```

```

while (i < 24) {
    timestampStart =
        year + "-" + month + "-" + day + " " + hour + ":" + minute + ":00"; // compose start timestamp

    timestampEnd = year + "-" + month + "-" + day + " " + hour + ":59:59"; //compose end timestamp

    calculateHourlyConsumption(
        timestampStart,
        timestampEnd,
        i,
        increaseI,
        graph
    );
    //convert hour to number and increase i if lesser than 23
    hourInt = parseInt(hour);
    if (hourInt !== 23) {
        hourInt++;
    } else {
        hourInt = 23;
    }
    //convert end hour to string
    endHour = hourInt.toString();
    if (endHour.length === 1) {
        endHour = "0" + endHour;
    }

    hour = endHour;
}

function calculateHourlyConsumption(
    timestampStart,
    timestampEnd,
    i,
    callback,

```

```

graph
) {
  fetch(
    "http://localhost:3000/statisticsHourlyTank" +
    activeTank +
    "?q1=" +
    timestampStart +
    "&q2=" +
    timestampEnd
  ).then((response) => {
    response.json().then((data) => {
      if (data.status === "notfound") {
        consumptionArray[i] = 0;
      } else {
        consumptionArray[i] = {
          hour: i.toString(),
          consumption: data.results.consumption,
        };
      }
      var flag = false;
      // check if consumption array is full with consumption values
      for (var j = 0; j < 24; j++) {
        flag = true;
        if (consumptionArray[j] === undefined) {
          flag = false;
        }
      }
      if (flag === true) {////////////////////if flag is true array is full draw bar////////////////////}
      var yLabelString;
      if (language === "english") {
        yLabelString = "↑ Consumption";
      } else if (language === "greek") {
        yLabelString = "↑ Κατανάλωση";
      }
      chart = BarChart(consumptionArray, {

```

```

        x: (d) => d.hour,
        y: (d) => d.consumption,
        yLabel: yLabelString,
        width: 600,
        height: 600,
        color: "rgb(62,208,255)",
    });
    $(".d3Button1").attr("disabled", false);
    $(".d3Button2").attr("disabled", false);
    }////////////////////////////////////
    }
    });
});

    callback();
}
//callback function to increase i
function increaseI() {
    i++;
}

function convertString(intNumber) {
    var str = intNumber.toString();
    if (str.length == 1) {
        str = "0" + str;
    }
    return str;
}
};

d3graphWeekly = (graph) => {
    changexAxis(graph);
    var titleOfDate=[];

```

```

consumptionArray = [];

var i = 0;

var selectedDate = $(".d3date1").prop("value");
var year = selectedDate.slice(0, 4);
var month = selectedDate.slice(5, 7);
var day = selectedDate.slice(8, 10);

var ds = parseInt(day);
var ms = parseInt(month);
var ys = parseInt(year);
var de, me, ye;
//chech if it is lead year
var leapYear = new Date(year, 1, 29).getMonth() == 1;

var daysOfMonth;

var timestampStart;
var timestampEnd;

var dayInt;
while (i < 7) {
    titleOfDate[i]= convertString(ds)+"/"+convertString(ms);
    ye = ys;
    me = ms;
    de = ds;

    timestampStart =
        convertString(ys) +
        "-" +
        convertString(ms) +
        "-" +
        convertString(ds) +

```

```
" " +  
"00:00:00";
```

```
timestampEnd =  
    convertString(ye) +  
    "-" +  
    convertString(me) +  
    "-" +  
    convertString(de) +  
    " " +  
    "23:59:59";
```

```
calculateHourlyConsumption(  
    timestampStart,  
    timestampEnd,  
    i,  
    increaseI,  
    graph  
);
```

```
if (ms == 2 && leapYear == false) { // if not leap year feb has 28  
    daysOfMonth = 28;  
} else if (ms == 2 && leapYear == true) {  
    daysOfMonth = 29; /// if not leap year feb has 28  
} else {  
    if (m31.includes(ms)) {  
        daysOfMonth = 31;  
    } else {  
        daysOfMonth = 30;  
    }  
}
```

```
if (ds + 1 > daysOfMonth) {  
    ds = 0;
```

```

    if (ms == 12) {
        ms = 0;
        ys++;
    }
    ms++;
}
ds++;
}

```

```

function calculateHourlyConsumption(
    timestampStart,
    timestampEnd,
    i,
    callback,
    graph
) {
    fetch(
        "http://localhost:3000/statisticsDailyTank" +
        activeTank +
        "?q1=" +
        timestampStart +
        "&q2=" +
        timestampEnd
    ).then((response) => {
        response.json().then((data) => {
            if (data.status == "notfound") {
                consumptionArray[i] = 0;
            } else {
                consumptionArray[i] = {
                    hour: titleOfDate[i],
                    consumption: data.results.consumption,
                };

                var flag = false;
            }
        });
    });
}

```

```

for (var j = 0; j < 7; j++) {
    flag = true;
    if (consumptionArray[j] == undefined) {
        flag = false;
    }
}
if (flag == true) {
    var yLabelString;
    if (language == "english") {
        yLabelString = "↑ Consumption";
    } else if (language == "greek") {
        yLabelString = "↑ Κατανάλωση";
    }
}

chart = BarChart(consumptionArray, {
    x: (d) => d.hour,
    y: (d) => d.consumption,
    yLabel: yLabelString,
    width: 600,
    height: 600,
    color: "rgb(62,208,255)",
});
$(".d3Button1").attr("disabled", false);
$(".d3Button2").attr("disabled", false);
}
});
});

callback();
}

function increaseI() {

```

```

    i++;
}

function convertString(intNumber) {
    var str = intNumber.toString();
    if (str.length == 1) {
        str = "0" + str;
    }
    return str;
}
};

```

Time.js

```
var m31 = [1, 3, 5, 7, 8, 10, 12];
```

```
var consumptionArray = [];
```

```
function BarChart(
    data,
    {
        x = (d, i) => i, // given d in data, returns the (ordinal) x-value
        y = (d) => d, // given d in data, returns the (quantitative) y-value
        title, // given d in data, returns the title text
        marginTop = 20, // the top margin, in pixels
        marginRight = 0, // the right margin, in pixels
        marginBottom = 30, // the bottom margin, in pixels
        marginLeft = 40, // the left margin, in pixels
        width = 640, // the outer width of the chart, in pixels
        height = 400, // the outer height of the chart, in pixels
        xDomain, // an array of (ordinal) x-values
        xRange = [marginLeft, width - marginRight], // [left, right]
        yType = d3.scaleLinear, // y-scale type
    }
) {
    // ...
}

```

```

yDomain, // [ymin, ymax]
yRange = [height - marginBottom, marginTop], // [bottom, top]
xPadding = 0.1, // amount of x-range to reserve to separate bars
yFormat, // a format specifier string for the y-axis
yLabel, // a label for the y-axis
color = "currentColor", // bar fill color
} = {}
) {
  // Compute values.
  const X = d3.map(data, x);
  const Y = d3.map(data, y);

  // Compute default domains, and unique the x-domain.
  if (xDomain === undefined) xDomain = X;
  if (yDomain === undefined) yDomain = [0, d3.max(Y)];
  xDomain = new d3.InternSet(xDomain);

  // Omit any data not present in the x-domain.
  const I = d3.range(X.length).filter((i) => xDomain.has(X[i]));

  // Construct scales, axes, and formats.
  const xScale = d3.scaleBand(xDomain, xRange).padding(xPadding);
  const yScale = yType(yDomain, yRange);
  const xAxis = d3.axisBottom(xScale).tickSizeOuter(0);
  const yAxis = d3.axisLeft(yScale).ticks(height / 40, yFormat);

  // Compute titles.
  if (title === undefined) {
    const formatValue = yScale.tickFormat(100, yFormat);
    title = (i) => `${X[i]}\n${formatValue(Y[i])}`;
  } else {
    const O = d3.map(data, (d) => d);
    const T = title;
    title = (i) => T(O[i], i, data);
  }
}

```

```
}
```

```
var previousSvg = document.getElementById("graph"); // check for a previous svg existence
```

```
if (previousSvg !== null) {
```

```
  previousSvg.remove();
```

```
}
```

```
// create svg and import attributes
```

```
const svg = d3
```

```
  .select("#d3-form-div")
```

```
  .append("svg")
```

```
  .attr("id", "graph")
```

```
  .attr("width", width)
```

```
  .attr("height", height)
```

```
  .attr("viewBox", [0, 0, width, height])
```

```
  .attr("style", "max-width: 100%; height: auto; height: intrinsic;");
```

```
svg
```

```
  .append("g")
```

```
  .attr("transform", `translate(${marginLeft},0)`)
```

```
  .call(yAxis)
```

```
  .call((g) => g.select(".domain").remove())
```

```
  .call((g) =>
```

```
    g
```

```
      .selectAll(".tick line")
```

```
      .clone()
```

```
      .attr("x2", width - marginLeft - marginRight)
```

```
      .attr("stroke-opacity", 0.1)
```

```
    )
```

```
  .call((g) =>
```

```
    g
```

```
      .append("text")
```

```
      .attr("x", -marginLeft)
```

```
      .attr("y", 10)
```

```
      .attr("fill", "currentColor")
```

```

        .attr("text-anchor", "start")
        .attr("font-size", 13)
        .text(yLabel)
    );
    // create svg bars
    const bar = svg
        .append("g")
        .attr("fill", color)
        .selectAll("rect")
        .data(I)
        .join("rect")
        .attr("x", (i) => xScale(X[i]))
        .attr("y", (i) => yScale(Y[i]))
        .attr("height", (i) => yScale(0) - yScale(Y[i]))
        .attr("width", xScale.bandwidth());
    if (title) bar.append("title").text(title);

    svg
        .append("g")
        .attr("transform", `translate(0,${height - marginBottom})`)
        .call(xAxis);

    return svg.node();
}

d3graphDaily = (graph) => {
    changexAxis(graph);

    consumptionArray = [];

    var i = 0;

    var selectedDate = $(".d3date1").prop("value"); // store the date from user selection
    var year = selectedDate.slice(0, 4);

```

```

var month = selectedDate.slice(5, 7);
var day = selectedDate.slice(8, 10);
var hour = "00";
var minute = "00";

var timestampStart;
var timestampEnd;
var f = 0;

var endHour;
var hourInt;
while (i < 24) {
    timestampStart =
        year + "-" + month + "-" + day + " " + hour + ":" + minute + ":00"; // compose start timestamp
    timestampEnd = year + "-" + month + "-" + day + " " + hour + ":59:59"; //compose end timestamp
    calculateHourlyConsumption(
        timestampStart,
        timestampEnd,
        i,
        increaseI,
        graph
    );
    //convert hour to number and increase i if lesser than 23
    hourInt = parseInt(hour);
    if (hourInt != 23) {
        hourInt++;
    } else {
        hourInt = 23;
    }
    //convert end hour to string
    endHour = hourInt.toString();
    if (endHour.length == 1) {
        endHour = "0" + endHour;
    }
}

```

```

    hour = endHour;
}
function calculateHourlyConsumption(
    timestampStart,
    timestampEnd,
    i,
    callback,
    graph
) {
    fetch(
        "http://localhost:3000/statisticsHourlyTank" +
        activeTank +
        "?q1=" +
        timestampStart +
        "&q2=" +
        timestampEnd
    ).then((response) => {
        response.json().then((data) => {
            if (data.status === "notfound") {
                consumptionArray[i] = 0;
            } else {
                consumptionArray[i] = {
                    hour: i.toString(),
                    consumption: data.results.consumption,
                };
                var flag = false;
                // check if consumption array is full with consumption values
                for (var j = 0; j < 24; j++) {
                    flag = true;
                    if (consumptionArray[j] === undefined) {
                        flag = false;
                    }
                }
            }
        })
    })
}

```

```

if (flag == true) {////////////////////if flag is true array is full drow bar////////////////////
    var yLabelString;
    if (language == "english") {
        yLabelString = "↑ Consumption";
    } else if (language == "greek") {
        yLabelString = "↑ Κατανάλωση";
    }
    chart = BarChart(consumptionArray, {
        x: (d) => d.hour,
        y: (d) => d.consumption,
        yLabel: yLabelString,
        width: 600,
        height: 600,
        color: "rgb(62,208,255)",
    });
    $(".d3Button1").attr("disabled", false);
    $(".d3Button2").attr("disabled", false);
}////////////////////
}
});
});

callback();
}
//callback function to increase i
function increaseI() {
    i++;
}

function convertString(intNumber) {
    var str = intNumber.toString();
    if (str.length == 1) {
        str = "0" + str;
    }
}

```

```

    return str;
  }
};

```

```

d3graphWeekly = (graph) => {
  changexAxis(graph);
  var titleOfDate=[];
  consumptionArray = [];
  var i = 0;

  var selectedDate = $(".d3date1").prop("value");
  var year = selectedDate.slice(0, 4);
  var month = selectedDate.slice(5, 7);
  var day = selectedDate.slice(8, 10);

  var ds = parseInt(day);
  var ms = parseInt(month);
  var ys = parseInt(year);
  var de, me, ye;
  //chech if it is lead year
  var leapYear = new Date(year, 1, 29).getMonth() == 1;
  var daysOfMonth;
  var timestampStart;
  var timestampEnd;
  var dayInt;
  while (i < 7) {
    titleOfDate[i]= convertString(ds)+"-"+convertString(ms);
    ye = ys;
    me = ms;
    de = ds;
    timestampStart =
      convertString(ys) +
      "-" +
      convertString(ms) +

```

```

    "-" +
    convertString(ds) +
    " " +
    "00:00:00";

timestampEnd =
    convertString(ye) +
    "-" +
    convertString(me) +
    "-" +
    convertString(de) +
    " " +
    "23:59:59";

calculateHourlyConsumption(
    timestampStart,
    timestampEnd,
    i,
    increaseI,
    graph
);

if (ms == 2 && leapYear == false) { // if not leap year feb has 28
    daysOfMonth = 28;
} else if (ms == 2 && leapYear == true) {
    daysOfMonth = 29; /// if not leap year feb has 28
} else {
    if (m31.includes(ms)) {
        daysOfMonth = 31;
    } else {
        daysOfMonth = 30;
    }
}
}

```

```

if (ds + 1 > daysOfMonth) {
  ds = 0;
  if (ms == 12) {
    ms = 0;
    ys++;
  }
  ms++;
}
ds++;
}
function calculateHourlyConsumption(
  timestampStart,
  timestampEnd,
  i,
  callback,
  graph
) {
  fetch(
    "http://localhost:3000/statisticsDailyTank" +
    activeTank +
    "?q1=" +
    timestampStart +
    "&q2=" +
    timestampEnd
  ).then((response) => {
    response.json().then((data) => {
      if (data.status == "notfound") {
        consumptionArray[i] = 0;
      } else {
        consumptionArray[i] = {
          hour: titleOfDate[i],
          consumption: data.results.consumption,
        };
      }
      var flag = false;
    });
  });
}

```

```

for (var j = 0; j < 7; j++) {
    flag = true;
    if (consumptionArray[j] == undefined) {
        flag = false;
    }
}
if (flag == true) {
    var yLabelString;
    if (language == "english") {
        yLabelString = "↑ Consumption";
    } else if (language == "greek") {
        yLabelString = "↑ Κατανάλωση";
    }
}

chart = BarChart(consumptionArray, {
    x: (d) => d.hour,
    y: (d) => d.consumption,
    yLabel: yLabelString,
    width: 600,
    height: 600,
    color: "rgb(62,208,255)",
});
$(".d3Button1").attr("disabled", false);
$(".d3Button2").attr("disabled", false);
}
});
});

callback();
}
function increaseI() {
    i++;

```

```
}  
function convertString(intNumber) {  
    var str = intNumber.toString();  
    if (str.length == 1) {  
        str = "0" + str;  
    }  
    return str;  
}  
};
```

Tooltip.js

```
$(".tank1").mouseover(function () {  
    showTooltip(1);  
});  
$(".tank1").mouseout(function () {  
    hideTooltip();  
});
```

```
$(".tank2").mouseover(function () {  
    showTooltip(2);  
});
```

```
$(".tank2").mouseout(function () {  
    hideTooltip();  
});
```

```
$(".tank3").mouseover(function () {  
    showTooltip(3);  
});
```

```
$(".tank3").mouseout(function () {
```

```

    hideTooltip();
});

$(".tank4").mouseover(function () {
    showTooltip(4);
});

$(".tank4").mouseout(function () {
    hideTooltip();
});

function showTooltip(tank) {
    // $(".tooltip").css("opacity",1);
    // $(".tooltip").css("text-align","center");
    var tankString="./assets/tank"+tank.toString()+".jpg";
    $(".imageTooltip").prop("src",tankString);

    if (tank > 2) {
        if (language == "english") {
            $(".tooltipText").html(
                "TANK: " +
                tanknames[tank - 1] +
                "<hr>" +
                "Current Volume: -" +
                "<br> Current Rate: -"
            );
        } else if (language == "greek") {
            $(".tooltipText").html(
                "ΔΕΞΑΜΕΝΗ: " +
                tanknamesgr[tank - 1] +
                "<hr>" +
                "Τρέχων Όγκος: -" +
                "<br>Τρέχουσα Τιμή Εισόδου: -"
            );
        }
    }
}

```

```

    }
} else {
    if (language == "english") {
        $(".tooltipText").html(
            "TANK: " +
            tanknames[tank - 1] +
            "<hr>" +
            "Current Volume: " +
            currentVolume[tank - 1].toFixed(3) +
            "<br> Current Rate: " +
            currentRate[tank - 1].toFixed(1)
        );
    } else if (language == "greek") {
        $(".tooltipText").html(
            "ΔΕΞΑΜΕΝΗ: " +
            tanknamesgr[tank - 1] +
            "<hr>" +
            "Τρέχων Όγκος: " +
            currentVolume[tank - 1].toFixed(3) +
            "<br> Τρέχουσα Τιμή Εισόδου: " +
            currentRate[tank - 1].toFixed(1)
        );
    }
}

$(".toolt").css("display", "block");
var coords = coordinates[[tank - 1]];
$(".toolt").css("left", coords[0] + "px");
$(".toolt").css("bottom", coords[1] + "px");
}

function hideTooltip() {
    $(".toolt").css("display", "none");
}

```

Language.js

```
var language = "english";
var numberOfElements = 22;
languageLabels = [
  ["Tanks", "Δεξαμενές"],
  ["Gastouri", "Γαστούρι"],
  ["Kavos", "Κάβος"],
  ["Sokraki", "Σωκράκι"],
  ["Ermones", "Ερμονες"],
  ["Options", "Επιλογές"],
  ["Set Alarm Levels", "Καθορισμός Επιπέδων Συναγερμού"],
  ["Set Language Greek", "Καθορισμός Αγγλικής Γλώσσας"],
  ["Submit", "Υποβολή"],
  ["Current Timestamp :", "Τρέχουσα Χρονοσφραγίδα :"],
  ["Max Level", "Μέγιστη Στάθμη"],
  ["Min Level", "Κατώτατη Στάθμη"],
  ["Filling Level", "Επίπεδο Πλήρωσης"],
  ["current Volume", "Τρέχων Όγκος"],
  ["Current Rate", "Τρέχουσα ροή Εισόδου"],
  ["Consumption Per Minute", "Κατανάλωση ανά Λεπτό"],
  ["Daily Consumption Graph", "Ημερήσια Κατανάλωση"],
  ["Weekly Consumption Graph", "Εβδομαδιαία Κατανάλωση"],
  ["Max Level", "Μέγιστη Τιμή"],
  ["Min Level", "Κατώτατη Τιμή"],
  ["Alarm Levels", "Επίπεδα Συναγερμού"],
  ["Max Volume of Tank", "Μέγεθος Δεξαμενής"]
];
elements = [
  "TanksMenu",
  "gastouri",
  "kavos",
  "sokraki",
```

```

    "ermones",
    "options",
    "alarms",
    "language",
    "submit",
    "currentTimestamp",
    "maxLevel",
    "minLevel",
    "fillingLevel",
    "currentVolume",
    "currentRate",
    "consumptionPerMinute",
    "dailyConsumptionGraph",
    "weeklyCosumptionGraph",
    "alarmMaxLevel",
    "alarmMinLevel",
    "alarmLevels",
    "tankMaxVolume"
];
var languageNumber = 0;
changeLanguage = () => {
    if (language == "english") {
        languageNumber = 1;
        language = "greek";
    } else if (language == "greek") {
        languageNumber = 0;
        language = "english";
    }

    for (var j = 0; j < numberOfElements; j++) {
        if (j < 12) {
            $("#" + elements[j]).text(languageLabels[j][languageNumber]);
        } else {
            $("#" + elements[j]).val(languageLabels[j][languageNumber]);
        }
    }
}

```

```

    }
}

changexAxis(d3Period);
checkLevels();
};

changexAxis = (graph) => {
    var xAxisString;

    if (language == "english") {
        if (graph == "weekly") {
            xAxisString = "Days →";
        } else if (graph == "daily") {
            xAxisString = "Hours →";
        }
    } else if (language == "greek") {
        if (graph == "weekly") {
            xAxisString = "Ημέρες →";
        } else if (graph == "daily") {
            xAxisString = "Ωρες →";
        }
    }
}

$(".x-axis").text(xAxisString);
};

```