

ΙΟΝΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ



Πτυχιακή Εργασία

ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ ΓΙΑ ΤΗΝ ΟΠΤΙΚΟΠΟΙΗΣΗ ΔΕΔΟΜΕΝΩΝ ΚΥΚΛΟΦΟΡΙΑΣ ΤΟΥ ΟΔΙΚΟΥ ΔΙΚΤΥΟΥ ΣΕ ΕΞΥΠΝΕΣ ΠΟΛΕΙΣ

Όνοματεπώνυμο Φοιτητή: Βογδόπουλος Γεώργιος ΑΜ: Π2018170

Επιβλέπουσα Καθηγήτρια: Χριστοπούλου Ελένη

Φεβρουάριος, 2024

Επιβλέπουσα Καθηγήτρια

Χριστοπούλου Ελένη, *ΕΔΙΠ*,
Ιόνιο Πανεπιστήμιο

Τριμελής Επιτροπή

Στεφανιδάκης Μιχαήλ, *Αναπληρωτής Καθηγητής*,
Ιόνιο Πανεπιστήμιο

Οικονόμου Κωνσταντίνος, *Καθηγητής*,
Ιόνιο Πανεπιστήμιο

Περίληψη

Στην εποχή της τεχνολογικής εξέλιξης, οι έξυπνες πόλεις αναδύονται ως καταλύτης για την ενίσχυση της αειφόρας και της αστικής ανάπτυξης. Τέτοιες πόλεις ενσωματώνουν συστήματα λογισμικού και διαδικτυακές υποδομές με στόχο να παρέχουν επίγνωση του πραγματικού κόσμου. Αυτό που κάνει μια έξυπνη πόλη λειτουργική είναι η συνδυασμένη χρήση προηγμένων τεχνολογιών υποδομής για την παροχή των βασικών υπηρεσιών της στο κοινό με εξαιρετικά αποτελεσματικό τρόπο. Η κατανόηση του οδικού δικτύου μιας πόλης είναι ζωτικής σημασίας στο σχεδιασμό της καθώς επηρεάζει άμεσα τόσο το βιοτικό επίπεδο των πολιτών της όσο και την ασφάλειά τους. Στην εργασία αυτή παρουσιάζεται ο σχεδιασμός και η υλοποίηση μιας εφαρμογής που απεικονίζει την κατάσταση στο οδικό δίκτυο. Για να το επιτύχουμε αυτό χρησιμοποιήθηκαν ανοιχτά δεδομένα που παρέχονται για την πόλη της Αθήνας. Οπτικοποιώντας τα δεδομένα αυτά μπορούμε να εξάγουμε γνώση η οποία μπορεί να χρησιμοποιηθεί τόσο από τις αρμόδιες αρχές που θα συμβάλει στην λήψη αποφάσεων όσο αφορά το οδικό δίκτυο, όσο και από τους πολίτες ώστε να κατανοήσουν καλύτερα τα μοτίβο της κυκλοφορίας στο οδικό δίκτυο της Αθήνας.

Λέξεις κλειδιά: Έξυπνη Πόλη, Ανοιχτά Δεδομένα, Οδικό Δίκτυο, Οπτικοποίηση Δεδομένων

Abstract

In the era of technological evolution, smart cities emerge as a catalyst for enhancing sustainable and urban development. Such cities incorporate software systems and internet infrastructures with the aim of providing real-world awareness. What makes a smart city functional is the combined use of advanced infrastructure technologies to provide its basic services to the public in an extremely efficient way. Understanding a city's road network is vital in its design as it directly affects both the standard of living of its citizens and their safety. This work presents the design and implementation of an application that depicts the state of the road network. To achieve this, open data provided for the city of Athens were used. By visualizing this data, we can extract knowledge which can be used both by the competent authorities that will contribute to decision-making regarding the road network, and by the citizens so as to better understand the traffic patterns on the road network of Athens.

Key Words: Smart City, Open Data, Road Network, Data Visualization

Ευχαριστίες

Μέσα από αυτή την εργασία θα ήθελα να ευχαριστήσω την επιβλέπουσα καθηγήτρια κυρία Ελένη Χριστοπούλου για την πολύτιμη συμβολή της στα έτη φοίτησής μου στο Ιόνιο Πανεπιστήμιο και για την καθοδήγηση και υποστήριξη που παρείχε καθ' όλη την διάρκεια της εκπόνησης της παρούσας Πτυχιακής Εργασίας.

Επίσης θα ήθελα να ευχαριστήσω την οικογένεια μου, την αδελφή μου και τους φίλους μου για την υποστήριξη που παρείχαν καθ' όλη τη διάρκεια των σπουδών μου.

Περιεχόμενα

A	Εισαγωγή	1
A.1	ΣΚΟΠΟΣ ΚΑΙ ΣΤΟΧΟΣ	2
A.2	ΣΥΝΕΙΣΦΟΡΑ	2
A.3	ΔΙΑΡΘΡΩΣΗ	2
B	Βιβλιογραφική Ανασκόπηση	5
B.1	ΕΝΤΟΠΙΣΜΟΣ ΠΡΟΒΛΗΜΑΤΟΣ	5
B.2	ΠΑΡΟΜΟΙΕΣ ΕΡΓΑΣΙΕΣ	6
B.3	ΟΡΙΣΜΟΙ	8
B.4	ΜΕΛΕΤΕΣ ΠΕΡΙΠΤΩΣΗΣ	9
C	Εργαλεία που χρησιμοποιήθηκαν	11
C.0.1	Spring Boot	11
C.0.2	Maven	12
C.0.3	Angular	13
C.0.4	Docker	14
C.1	ΠΑΡΟΥΣΙΑΣΗ ΕΦΑΡΜΟΓΗΣ	15
C.1.1	Dashboard	15
C.1.2	Data Overview	23
C.1.3	API Info	28
D	Υλοποίηση Εφαρμογής	30
D.1	BACKEND	30
D.1.1	Spring Boot	31

D.1.1.1	<i>Αρχιτεκτονική Backend</i>	31
D.1.1.2	<i>Dependency Management</i>	33
D.1.1.3	<i>Συλλογή δεδομένων</i>	37
D.1.1.4	<i>Διαχείριση και επεξεργασία δεδομένων</i>	41
D.1.1.5	<i>Εγγραφή δεδομένων σε XML</i>	46
D.1.1.6	<i>Εγγραφή δεδομένων σε JSON</i>	49
D.1.1.7	<i>Εγγραφή δεδομένων σε Excel</i>	50
D.1.1.8	<i>Σχεδιασμός API</i>	54
D.2	FRONTEND	61
D.2.1	Angular και Angular Material	61
D.2.1.1	<i>Αρχιτεκτονική Frontend</i>	61
D.2.1.2	<i>Σχεδιασμός Services</i>	62
D.2.1.3	<i>Σχεδιασμός Components</i>	66
D.2.1.4	<i>Οπτικοποίηση Δεδομένων</i>	74
D.3	CONTAINERIZATION	79
D.3.1	Docker και Docker Compose	79
D.3.1.1	<i>Dockerfile</i>	79
D.3.1.2	<i>Docker Compose</i>	80
E	Συμπεράσματα και Μελλοντικές κατευθύνσεις	82
E.1	ΣΥΜΠΕΡΑΣΜΑΤΑ	82
E.2	ΜΕΛΛΟΝΤΙΚΕΣ ΚΑΤΕΥΘΥΝΣΕΙΣ	83
	Συντμήσεις	88
	Γλωσσάρι Ξενικών Όρων	90

Κατάλογος Σχημάτων

C.1	Αρχική οθόνη εφαρμογής	16
C.2	Ημερολόγιο για την επιλογή της χρονικής περιόδου	16
C.3	Animation αφού έχει επιλεγεί η χρονική περίοδος	17
C.4	Dashboard μετά την επιλογή ημερομηνίας για την οπτικοποίηση δεδομένων .	18
C.5	Αναπαράσταση δεδομένων κυκλοφορίας σχετικά με τον αριθμό οχημάτων . .	19
C.6	Αναπαράσταση δεδομένων κυκλοφορίας σχετικά με την μέση ταχύτητα των οχημάτων	20
C.7	Διαγράμματα για τον αριθμό των αυτοκινήτων	21
C.8	Διαγράμματα για την ταχύτητα των οχημάτων	21
C.9	Dashboard για τους δρόμους με μέτρια έως και λίγη κίνηση	21
C.10	Μήνυμα μη διαθέσιμων δεδομένων από την εφαρμογή	22
C.11	Παρουσίαση ενότητας Data Overview	23
C.12	Animation εφαρμογής	24
C.13	Αναδυόμενο ημερολόγιο	24
C.14	Data Overview επιλογή χρονικής περιόδου και ανάκτηση δεδομένων	24
C.15	Εμφάνιση δεδομένων εφαρμογής και παρουσίαση δυνατότητας φιλτραρίσματος και ταξινόμησης τους	25
C.16	Εμφάνιση φιλτραρισμένων δεδομένων για την περιοχή Κατεχάκη	26
C.17	Μένου για την λήψη των δεδομένων σε αρχεία	27
C.18	Αποτέλεσμα λήψης αρχείου Excel	27
C.19	Αποτέλεσμα αρχείου XML	28
C.20	Αποτέλεσμα αρχείου JSON	28
C.21	Παρουσίαση επεξήγησης API endpoints μέσω της εφαρμογής	29
D.22	Αρχιτεκτονική Backend εφαρμογής	32

D.23 Αναπαράσταση του κώδικα <i>Backend</i> σε διάγραμμα	33
D.24 Αρχιτεκτονική <i>Angular Frontend</i>	62

Κατάλογος Πινάκων

<i>C.1 Σύγκριση Spring και Spring Boot.</i>	12
<i>C.2 Χαρακτηριστικά του Maven</i>	13
<i>D.1 Components εφαρμογής</i>	66

Κατάλογος Αναφορών Πυγαίου

Κώδικα

D.1 Περιγραφή του pom	34
D.2 Περιγραφή των sub-modules και των properties	34
D.3 Dependencies εφαρμογής	36
D.4 Διαμόρφωση του τρόπου χησίματος της εφαρμογής	37
D.5 Backend Web Client για την συλλογή δεδομένων από API	38
D.6 Κλάση που περιγράφει τα δεδομένα της εφαρμογής	39
D.7 Υπολογισμός γενικών πληροφοριών κίνησης	42
D.8 Ταξινόμηση δεδομένων κυκλοφορίας	44
D.9 Φιλτράρισμα δεδομένων ώστε να κρατήσουμε τις δέκα μεγαλύτερες εγγραφές	44
D.10 Φιλτράρισμα δεδομένων ώστε να κρατήσουμε τις μικρότερες εγγραφές	45
D.11 Δημιουργία Constructor και μεθόδου για την συλλογή δεδομένων στην Overview-TrafficWriteToXML	46
D.12 Δημιουργία Document για την εγγραφή του XML	47
D.13 Προσθήκη δεδομένων στο Measurement	47
D.14 Εγγραφή XML αρχείου	48
D.15 Εγγραφή σε αρχείο JSON	49
D.16 Δημιουργία workbook και style των κελιών	51
D.17 Δημιουργία Header υπολογιστικού φύλλου	51
D.18 Προσθήκη δεδομένων στα κελιά	53
D.19 Εγγραφή αρχείου XLSX	54
D.20 Controller για την λήψη όλων των δεδομένων κυκλοφορίας	55
D.21 Endpoint αποστολή του XML αρχείου	56
D.22 Endpoint αποστολή του JSON αρχείου	57

D.23 Endpoint αποστολή του Excel αρχείου	57
D.24 Endpoint για τα δεδομένα στα mini cards	59
D.25 Ταξινόμηση δεδομένων από τον WebClient	59
D.26 Αποστολή δεδομένων για την μεγαλύτερη προέλευση οχημάτων	60
D.27 Αποστολή δεδομένων για την μεσαία χαμηλή προέλευση οχημάτων	60
D.28 Dashboard Service για την επικοινωνία με το Backend	63
D.29 Table Service για την επικοινωνία με το Backend	65
D.30 Navigation Componet για την δημιουργία δυναμικού μενού	67
D.31 Δημιουργία αναδυόμενου ημερολογίου	68
D.32 Μέθοδοι για το format του input του ημερολογίου	69
D.33 Κλήση του service για την συλλογή δεδομένων του πίνακα	70
D.34 Κλήση του service για την συλλογή δεδομένων του πίνακα	70
D.35 Μέθοδος για το φιλτράρισμα των δεδομένων του πίνακα	72
D.36 Φίλτρο πίνακα	72
D.37 Μέθοδος για το κατέβασμα του αρχείου	73
D.38 Κλήση της downloadFile	73
D.39 Δημιουργία Bar Chart	74
D.40 Εμφάνιση του Bar Chart	76
D.41 Συλλογή δεδομένων για τα διαγράμματα υψηλής κίνησης	77
D.42 Συλλογή δεδομένων για τα διαγράμματα μέτριας έως και χαμηλής κίνησης	78
D.43 Spring Boot Backend Dockerfile	80
D.44 Angular Frontend Dockerfile	80
D.45 Docker Compose για την διαχείριση πολλαπλών container	81

Κεφάλαιο Α

Εισαγωγή

Σε ένα ραγδαία αναπτυσσόμενο κόσμο καθώς ο πληθυσμός των πόλεων αυξάνεται και οι πόροι γίνονται ολοένα και πιο περιορισμένοι, η ανάγκη για πιο αποτελεσματικές και βιώσιμες λύσεις γίνεται επιτακτική. Εδώ εμφανίζονται οι έξυπνες πόλεις, οι οποίες χρησιμοποιούν την τεχνολογία για να βελτιώσουν την ποιότητα ζωής των κατοίκων τους, να εξοικονομήσουν πόρους και να επιτύχουν βιώσιμη ανάπτυξη. Οι πόλεις αυτές χρησιμοποιούν την τεχνολογία τόσο σε φυσικό επίπεδο, με την χρήση διάφορων αισθητήρων όσο και συστήματα λογισμικού με στόχο να εξάγουν γνώση που θα αποτελεί σημαντικό εργαλείο στην καταπολέμηση αρκετών προβλημάτων σε ένα αστικό κέντρο.

Κατά τον σχεδιασμό μιας αστικής κοινότητας ένας από τους πιο σημαντικούς παράγοντες αποτελεί και το οδικό της δίκτυο. Ο καλός σχεδιασμός και διαχείριση του, είναι ζωτικής σημασίας καθώς άμεσα επηρεάζει την μεταφορά των κατοίκων, την μεταφορά των αγαθών, συμβάλει στην οικονομική ανάπτυξη και βιωσιμότητα. Καθώς οι πόλεις και η τεχνολογία εξελίσσονται προκύπτει όλο ένα και περισσότερο η ανάγκη για την εφαρμογή μέτρων τα οποία θα μας βοηθήσουν να κατανοήσουμε και να βελτιώσουμε το οδικό δίκτυο. Τέτοια μέτρα θα μας βοηθήσουν να εξάγουμε πληροφορία από πραγματικές συνθήκες η οποία θα βοηθήσει στην αντιμετώπιση των προβλημάτων κυκλοφορίας, την βελτίωση της αποδοτικότητας των δημόσιων μεταφορών, την μείωση των καθυστερήσεων και την βελτίωση της ασφάλειας των οδηγών. Επιπλέον η συλλογή και διάθεση των δεδομένων αυτών μπορεί να οδηγήσει στην δημιουργία καινοτόμων εφαρμογών που θα ενισχύσουν την αποτελεσματικότητα του.

Επομένως, η έξυπνη πόλη αποτελεί έναν ζωντανό οργανισμό που ζει, δρα και αλληλοεπιδρά με άλλους οργανισμούς στα πλαίσια αλληλοβοήθειας και προοπτικής για μελλοντική ανάπτυξη [29]

A.1 Σκοπός και Στόχος

Σκοπός της συγκεκριμένης εργασίας είναι ο σχεδιασμός και η ανάπτυξη μιας εφαρμογής για την οπτικοποίηση δεδομένων κυκλοφορίας στο οδικό δίκτυο. Τα δεδομένα αυτά θα παρέχονται από ένα API της κυβέρνησης και η εφαρμογή θα τα επεξεργάζεται με σκοπό να εξάγουμε κάποια γνώση, να τα οπτικοποιεί σε διάφορα διαγράμματα και να μπορούμε να δημιουργούμε διάφορα αρχεία βάση αυτών για να τα παρέχουμε στο χρήστη για περαιτέρω επεξεργασία. Στόχος αυτού είναι να καταστήσουμε τα δεδομένα πιο κατανοητά και προσβάσιμα, επιτρέποντας στους χρήστες να αναλύουν και να ερμηνεύουν τα μοτίβα κυκλοφορίας με μεγαλύτερη ακρίβεια και αποτελεσματικότητα.

A.2 Συνεισφορά

Η συνεισφορά αυτής της εργασίας είναι πολυδιάστατη. Αρχικά η εφαρμογή αποτελεί μια ισχυρή βάση για την κατανόηση της κατάστασης στο οδικό δίκτυο παρέχοντας πληροφορία με διαφορετικούς τρόπους. Με την οπτικοποίηση των δεδομένων ο χρήστης μπορεί να δει εύκολα και γρήγορα δεδομένα από το οδικό δίκτυο μέσα από έναν εύχρηστο τρόπο. Με την διάθεση των δεδομένων αυτών σε αρχεία κάνουμε λόγο για διαθέσιμα και προσβάσιμα δεδομένα. Τα δεδομένα αυτά μπορούν να χρησιμοποιηθούν από αναλυτές και σχεδιαστές οδικών δικτύων για περαιτέρω μελέτη με στόχο την βελτίωση της αποτελεσματικότητας των οδικών δικτύων και της ανάπτυξης πιο βιώσιμων στρατηγικών μεταφοράς μέσω της κατανόησης της επίδρασης της κυκλοφορίας στο περιβάλλον

A.3 Διαρθρωση

Στο επόμενο κεφάλαιο ακολουθεί η βιβλιογραφική ανασκόπηση της εργασίας. Μέσα από αυτό το κεφάλαιο αναλύονται τα βασικά προβλήματα που παρατηρούνται στο οδικό δίκτυο σήμερα καθώς και τι αρνητικές επιπτώσεις φέρουν τόσο στα αστικά κέντρα όσο και στο βι-

οτικό επίπεδο των πολιτών. Στην συνέχεια εξετάζονται παρόμοιες εργασίες και έρευνες που έχουν πραγματοποιηθεί, μελετάμε τις προσεγγίσεις τους καθώς και τα αποτελέσματα που έχουν φέρει. Έπειτα μελετάμε τους βασικούς ορισμούς των Έξυπνων Πόλεων, Internet of Things, Intelligent Transport System και Smart Transportation. Κλείνοντας το κεφάλαιο αναφερόμαστε σε μελέτες περιπτώσεις όπου βλέπουμε κάποια παραδείγματα από συστήματα που χρησιμοποιούνται αυτή την στιγμή, αναφέρουμε τον τρόπο λειτουργίας τους και τα θετικά που έχουν φέρει.

Στο τρίτο κεφάλαιο γίνεται μια περιγραφή των εργαλείων που χρησιμοποιήθηκαν για τον σχεδιασμό και την ανάπτυξη της εφαρμογής. Τα εργαλεία αυτά αποτελούν το Spring Boot, το Maven, την Angular και το Docker. Αφού περιγράψουμε τα εργαλεία αυτά και αναφέρουμε την χρησιμότητα και λειτουργικότητα τους θα εξετάσουμε την εφαρμογή που αναπτύχθηκε. Θα δούμε τις διάφορες λειτουργίες της καθώς θα εξετάσουμε και την χρησιμότητα της μέσα από παραδείγματα. Στην αρχή παρουσιάζεται το Dashboard το οποίο αποτελεί και το τμήμα της εφαρμογής που είναι υπεύθυνο για τις οπτικοποιήσεις των δεδομένων, έπειτα ακολουθεί το Data Overview το σημείο στο οποίο μπορούμε να εξετάσουμε τα δεδομένα μας πιο αναλυτικά και να τα κατεβάσουμε σε αρχεία και τέλος το API Info το οποίο μας παρέχει ένα αναλυτικό API Documentation για τα Rest API του Backend.

Στο τέταρτο κεφάλαιο γίνεται η περιγραφή της υλοποίησης της εφαρμογής. Αρχικά ξεκινάμε με το Backend, αφού γίνει μια εισαγωγή στην έννοια του και ποιος είναι ο ρόλος του στην ανάπτυξη μιας εφαρμογής θα δούμε κάποια χαρακτηριστικά του Spring Boot.

Η περιγραφή του Backend ξεκινάει με την ανάλυση της αρχιτεκτονικής του, όπου μελετάμε τα διάφορα επίπεδα της Backend εφαρμογής. Συνεχίζοντας αναλύουμε τον τρόπο με τον οποίο διαχειρίστηκαν οι εξαρτήσεις της εφαρμογής μέσα από το Maven καθώς και τον τρόπο με τον οποίο γίνεται η λήψη των δεδομένων από την εφαρμογή μέσα από το API της Κυβέρνησης. Έπειτα μελετάμε τον τρόπο με τον οποίο επεξεργαζόμαστε τα δεδομένα αυτά με στόχο να εξάγουμε γνώση που θα χρησιμοποιεί η εφαρμογή στα διάφορα τμήματα της. Εξετάζουμε τον τρόπο με τον οποίο γίνεται η εγγραφή των δεδομένων σε αρχεία και τον σχεδιασμό των Rest API που θα ανταλλάσσουν πληροφορία με τον Frontend.

Στην συνέχεια ακολουθεί η περιγραφή της Frontend εφαρμογής. Γίνεται μια περιγραφή στο framework της Angular σε συνδυασμό με την Angular Material και της ChartJS που χρησι-

μοποιήθηκε για τις οπτικοποιήσεις. Έπειτα ακολουθεί η ανάλυση της αρχιτεκτονικής της frontend εφαρμογής καθώς βλέπουμε και τον τρόπο με τον οποίο ανταλλάσσει πληροφορία με το Backend. Παρακάτω αναλύουμε τα σχεδιασμό των Components και Services καθώς βλέπουμε και τον τρόπο με το οποίο παρουσιάζονται τα γραφήματα με τα δεδομένα του Backend. Καθώς τελειώνει το τέταρτο κεφάλαιο θα μελετήσουμε τον τρόπο με το οποίο οι δύο αυτές εφαρμογές οργανώθηκαν σε διάφορα containers καθώς και θα δούμε τι θετικά μπορεί να επιφέρει η συγκεκριμένη προσέγγιση

Στο πέμπτο κεφάλαιο, παρουσιάζονται τα κύρια συμπεράσματα που προέκυψαν από την εφαρμογή, καθώς και οι προτεινόμενες μελλοντικές κατευθύνσεις για την περαιτέρω ανάπτυξη και βελτίωση της.

Κεφάλαιο Β

Βιβλιογραφική Ανασκόπηση

Η συλλογή και ανάλυση δεδομένων κυκλοφορίας από μια πόλη αποτελεί έναν ισχυρό μηχανισμό για την κατανόηση της κατάστασης της κυκλοφορίας και των πιθανών προβλημάτων που μπορεί να παρουσιάζει. Στο επόμενο κεφάλαιο, θα εξετάσουμε διάφορες έρευνες που έχουν πραγματοποιηθεί σε αυτόν τον τομέα, επικεντρώνοντας στα αποτελέσματα και τις επιπτώσεις τους στην κυκλοφορία. Θα αναλύσουμε τους βασικούς ορισμούς και θα παρουσιάσουμε μελέτες περίπτωσης από συστήματα που χρησιμοποιούνται σήμερα για την βελτίωση της κυκλοφορίας στις πόλεις. Αυτή η ανάλυση θα μας βοηθήσει να κατανοήσουμε καλύτερα τις τάσεις και τις προκλήσεις σε αυτόν τον τομέα.

B.1 Εντοπισμός Προβλήματος

Το οδικό δίκτυο μέσα σε ένα αστικό κέντρο αποτελεί από τους πιο σημαντικούς παράγοντες καθώς μέσω αυτού γίνεται η μεταφορά αγαθών και πολιτών, συνδέει τα διάφορα τμήματα της πόλης και συμβάλει στο τουρισμό και την ανάπτυξη της περιοχής. Ωστόσο παρά τον σημαντικό ρόλο που διαδραματίζει μπορούμε να εντοπίσουμε αρκετά προβλήματα. Τα προβλήματα στην κυκλοφορία, οι ανεπαρκείς υποδομές, το μεγάλο κόστος για την συντήρησή τους και η μειωμένη ασφάλεια οδικής κυκλοφορίας αποτελούν κάποια από αυτά.

Σημαντική είναι η αύξηση τα τελευταία έτη της κυκλοφοριακής συμφόρησης. Από δεδομένα της Eurostat παρατηρείται αύξηση της χρήσης ιδιωτικών οχημάτων, όπου για το 2021 καταγράφηκε αναλογία 1.000 οχήματα ανά κάτοικο [9]. Τέτοια δεδομένα αναφέρονται κυρίως σε

πόλεις της βόρειας Ιταλίας, πόλεις του Φιλανδικού αρχιπελάγους καθώς και στην περιοχή της Αττικής στην Αθήνα. Παράγοντας που συμβάλει στην αύξηση της κυκλοφοριακής συμμόρφωσης. Η αύξηση αυτή παρατηρείται από τις αρχές της πανδημίας του COVID-19, όπου ο αριθμός των διεθνών ταξιδιωτών παραμένει χαμηλός και οι άνθρωποι χρησιμοποιούν λιγότερο τα Μέσα Μαζικής Μεταφοράς για τις μετακινήσεις τους, ωστόσο η αυξημένη χρήση των ιδιωτικών οχημάτων παρατηρείται ακόμα και μετά το τέλος της πανδημίας [9].

Τα προβλήματα στο οδικό δίκτυο φέρουν επιπτώσεις τόσο στο περιβάλλον όσο και στους πολίτες των αστικών κέντρων. Η αυξημένη κυκλοφοριακή συμμόρφωση μπορεί να συμβάλει στην αύξηση των εκπομπών διοξειδίου του άνθρακα που μπορεί να επηρεάσει τόσο το βιοτικό επίπεδο όσο και την βιωσιμότητα σε ένα αστικό κέντρο. Ταυτόχρονα ανησυχητική αποτελεί και η αύξηση των τροχαίων ατυχημάτων τα τελευταία έτη επιδεικνύοντας μας την ανάγκη για εφαρμογή πιο επιτακτικών λύσεων όσο αφορά την δημιουργία, παρακολούθηση και συντήρηση των οδικών δικτύων. Σύμφωνα με δεδομένα της Ευρωπαϊκής Ένωσης σημαντική ήταν και η αύξηση των τροχαίων ατυχημάτων. Ο αριθμός των θανάτων από τροχαία ατυχήματα αυξήθηκε κατά 6 % το 2021 σε σύγκριση με το 2020 και 3% το 2022 σε σχέση με το 2021 [17].

B.2 Παρόμοιες Εργασίες

Σε έρευνα που δημοσιεύθηκε από το IEEE Internet of Things Journal [27] με σχετικό τίτλο “Capsule network assisted IoT traffic classification mechanism for smart cities”, προτείνεται η δημιουργία και ανάπτυξη ενός IoT (Internet of Things) μηχανισμού για την συλλογή και επεξεργασία δεδομένων κίνησης. Ο μηχανισμός αυτός θα ακολουθεί μια λογική κάψουλας που θα συλλέγει και θα ταξινομεί τα δεδομένα που συλλέγονται βάση ενός μοντέλου. Τα πειραματικά ευρήματα αυτού του συστήματος έδειξαν ότι η χρήση δικτύων κάψουλας (capsule networks) μπορεί να παρέχει ταξινομήσεις υψηλής ακρίβειας που σίγουρα συμβάλουν στην αποτελεσματικότητα του συστήματος. Από την άλλη πλευρά μια τέτοια προσέγγιση απαιτεί υψηλό κόστος μέτρησης και εκπαίδευσης του συστήματος.

Παράλληλα από έρευνα του Πανεπιστημίου του Perking στην Κίνα [4] σχετικά με την ανάπτυξη ενός Intelligent Transportation System (ITS) για την πρόβλεψη της κυκλοφοριακής συμμόρφωσης παρατηρήθηκε πως η χρήση long-term memory δικτύων μπορεί να μας παρέχει ακριβείς προβλέψεις οι οποίες ξεπερνάν σε απόδοση τα βασικά μοντέλα νευρωνικών δικτύων.

Ωστόσο οι δοκιμές πραγματοποιήθηκαν σε ένα περιορισμένο δίκτυο δρόμων με περιορισμένο αριθμό κόμβων δικτύου.

Σε παρόμοια έρευνα του πανεπιστήμιου της Modena της Ιταλίας [6] σχετικά με την ανάλυση κυκλοφοριακών δεδομένων για τις έξυπνες πόλεις υλοποιήθηκε ένα micro simulation μοντέλο το οποίο λάμβανε και επεξεργαζόταν δεδομένα κίνησης από διάφορους αισθητήρες που ήταν διατεταγμένοι στην πόλη. Στόχος ήταν να παρέχουν προβλέψεις σχετικά με το πως θα κυμανθεί η κίνηση στους δρόμους καθώς και να παρέχουν συγκρίσεις για το πως κυμάνθηκε μεταξύ των εργάσιμων ημερών και μη. Επιπλέον μελετήθηκαν δεδομένα για ημέρες αργιών και ημέρες που είχαν προειδοποίηση για καιρικά φαινόμενα. Μέσα από αυτή την έρευνα παρατηρήθηκε πως η κίνηση για τις εργάσιμες μέρες ακολουθούν παρόμοιο μοτίβο χωρίς να διαφέρουν πολύ μεταξύ τους. Αρκετές διαφοροποιήσεις στα μοτίβο κίνησης στο οδικό δίκτυο της Modena είχαν οι μέρες του Σαββάτου, της Κυριακής και οι αργίες καθώς συγκεκριμένοι αισθητήρες παρείχαν καταγραφές οχημάτων που ξεπερνούσαν τον μέσο όρο. Τα δεδομένα που χρησιμοποιήθηκαν σε αυτή την έρευνα κάλυπταν μόνο τα οχήματα που κινούνταν στο οδικό δίκτυο χωρίς να συμπεριλαμβάνονται άλλα μέσα μαζικής μεταφοράς όπως μετρό, ποδήλατα ακόμα και πεζοί.

Σε έρευνα που πραγματοποιήθηκε σχετικά με την ταξινόμηση δεδομένων οδικού δικτύου για έξυπνες πόλεις παρατηρήθηκε πως η χρήση του αλγορίθμου Decision Tree μπορεί να επιτύχει υψηλά ποσοστά ακρίβειας. Μέσα από αυτή την έρευνα μελετήθηκε η απόδοση τεσσάρων αλγορίθμων μηχανικής μάθησης ως προς τις ταξινομήσεις που παρέχουν για δεδομένα κίνησης στο οδικό δίκτυο. Οι αλγόριθμοι αυτοί αποτελούν τους Support Vector Machine, Decision Tree, K-nearest Neighbor και Random Forest. Μεγαλύτερα ποσοστά ακρίβειας έφερε ο αλγόριθμος του Decision Tree με ακρίβεια 99.18% ενώ αλγόριθμοι όπως ο KNN πέτυχε τα χαμηλότερα ποσοστά με ακρίβεια 97.16% [22].

Τα θετικά που μπορεί να επιφέρει η οπτικοποίηση δεδομένων κυκλοφορίας του οδικού δικτύου σε πραγματικό χρόνο μας παρουσιάζει μια έρευνα που διεξήχθη από το Εθνικό Πανεπιστήμιο Μόσχας. Στην συγκεκριμένη έρευνα αντλήθηκαν δεδομένα από αισθητήρες που ήταν διατεταγμένοι στο οδικό δίκτυο και έπειτα μέσα από κάποιο μοντέλο προσομοίωσης έγινε η εξαγωγή της πληροφορίας. Την πληροφορία αυτή την χρησιμοποίησαν στο να λάβουν αποφάσεις για τον χρόνο αναμονής στα φανάρια της πόλης και στην προβολή σημάτων στο οδικό δίκτυο. Αποτέλεσμα αυτού ήταν ο συνωστισμός σε μεγάλους δρόμους να μειωθεί κατά 5,56 %

και 2,78 % σε κεντρικούς δρόμους, αυτό οφείλετε κυρίως στις σημάνσεις που παρείχαν στον δρόμο. Μια άλλη παρατήρηση που έγινε ήταν αυτή της αύξησης του μέσου όρου ταχύτητας που ανέπτυσαν οι οδηγοί. Σε κεντρικούς δρόμους παρατηρήθηκε πως οι οδηγοί πλέον κινούνταν κατά μέσο όρο με 28 χλμ. ενώ πριν ο μέσος όρος ταχύτητας είχε καταγραφεί στα 25 χλμ. Σε μεγάλους αυτοκινητόδρομους ο μέσος όρος ταχύτητάς αυξήθηκε στα 50 χλμ. σε σχέση με τα 45 χλμ. που ήταν πριν και στις κατοικήσιμες περιοχές οι οδηγοί κινούνταν με 32 χλμ. σε σχέση με τα 30 χλμ. που ήταν ο προηγούμενος μέσος όρος [8].

B.3 Ορισμοί

Οι έξυπνες πόλεις αποτελούν αστικές περιοχές όπου με την χρήση της τεχνολογίας και της συλλογής δεδομένων μέσα από αυτή συμβάλουν στην βελτίωση του βιοτικού επιπέδου, της βιωσιμότητας και στην αποτελεσματική λειτουργία των υπηρεσιών που μπορεί να προσφέρει. Όπως το νευρικό σύστημα καθορίζει πώς οι άνθρωποι αντιδρούν στον κόσμο γύρω τους, έτσι και οι εξελισσόμενες τεχνολογίες εξουσιοδοτούν τις πόλεις να αντιδρούν στις αλλαγές στο τοπικό τους αστικό περιβάλλον. Η IBM ορίζει ως έξυπνη πόλη αυτή η οποία κάνει βέλτιστη χρήση όλων των διασυνδεδεμένων πληροφοριών που είναι διαθέσιμες σήμερα για να κατανοήσει και να ελέγχει καλύτερα τις λειτουργίες της και να βελτιστοποιεί τη χρήση των περιορισμένων πόρων [1]. Τέτοιες πόλεις χρησιμοποιούν τεχνολογίες όπως Internet of Things (IoT), τεχνητή νοημοσύνη (AI) και Information and Communication Technology (ICT) για να πετύχουν υψηλότερα επίπεδα βιωσιμότητας [24].

Ο όρος “Internet of Things” (IoT) δημιουργήθηκε από τον Kevin Ashton κατά τη διάρκεια μιας παρουσίασης στην Procter & Gamble το 1999. Ως τεχνολογίας οραματίζεται να συνδυάσει καταναλωτικά προϊόντα, ανθεκτικά αγαθά, οχήματα, βιομηχανικά εξαρτήματα, εγκαταστάσεις, αισθητήρες και άλλα καθημερινά αντικείμενα με το διαδίκτυο και την ανάλυση δεδομένων με στόχο να μεταμορφωθεί ο τρόπος που ζούμε και εργαζόμαστε [18]. Μέσα από αυτό το δίκτυο των πραγμάτων μπορούμε να εξάγουμε γνώση βάση του πραγματικού κόσμου που θα απαιτεί ελάχιστη ανθρώπινη παρέμβαση καθώς οι αισθητήρες και γενικά τα “αντικείμενα” θα επικοινωνούν μεταξύ τους [14].

Ο όρος Intelligent Transport Systems (ITS) αναφέρετε σε μια σειρά τεχνολογιών που σχεδιάστηκαν για να βελτιώσουν την ασφάλεια, την αποδοτικότητα και την περιβαλλον-

τολογική απόδοση των συστημάτων μεταφορών μέσα σε ένα αστικό κέντρο [28]. Με την χρήση διάφορων τεχνολογιών όπως το IoT τα έξυπνα συστήματα μεταφορών μπορούν να συμβάλουν στην μείωση της κυκλοφοριακής συμφόρησης, την μείωση εκπομπών διοξειδίου του άνθρακα στην ατμόσφαιρα και στην αποτελεσματικότητα της κυκλοφορίας μέσα σε ένα αστικό κέντρο [7]. Αυτό το πετυχαίνουν με το να συλλέγουν και να επεξεργάζονται δεδομένα σε πραγματικό χρόνο.

Ο όρος Smart Transportation ή Smart Mobility αναφέρονται σε συστήματα τα οποία αποσκοπούν στην ενορχήστρωση του τρόπου με τον οποίο τα οχήματα και οι άνθρωποι μετακινούνται σε μια πόλη. Αυτό πραγματοποιείται με την χρήση προηγμένων τεχνολογιών όπως το Internet of Things και γεωγραφικά δεδομένα μέσα από κινητές συσκευές [19]. Με την χρήση τέτοιων τεχνολογιών στοχεύουμε στην βελτίωση της κυκλοφορίας σε ένα αστικό κέντρο τόσο από τα οχήματα όσο και από τους πεζούς καθώς και στην γενική ασφάλεια τους [10].

B.4 Μελέτες Περίπτωσης

Παράδειγμα χρήσης συστημάτων για την παρακολούθηση της κατάστασης στο οδικό δίκτυο αποτελεί η Σιγκαπούρη [15], μέσο της ευρύτερης προσπάθειας της χώρας να γίνει το πρώτο έξυπνο έθνος. Με την ανάπτυξη και χρήση ενός προηγμένου Έξυπνου Συστήματος Μεταφορών (ITS) παρέχει λύσεις για την αντιμετώπιση της αστικής πυκνότητας. Το σύστημα αυτό ενσωματώνει μια σειρά από έξυπνες τεχνολογίες μεταφορών, συμπεριλαμβανομένου ενός από τα πρώτα συστήματα ηλεκτρονικής τιμολόγησης δρόμων στον κόσμο, πληροφορίες σε πραγματικό χρόνο μέσω οχημάτων που παρέχουν πληροφορία μέσα από GPS και μια ολοκληρωμένη δομή δημόσιων μεταφορών. Επιπλέον το σύστημα αυτό χρησιμοποιεί αισθητήρες, συστήματα αυτόματου ελέγχου κυκλοφορίας και ανάλυσης δεδομένων με στόχο να μεγιστοποιεί την αποδοτικότητα του οδικού δικτύου και να καταστήσει τους δρόμους ασφαλέστερους [15].

Η Ιαπωνία με στόχο την βελτίωση της ασφάλειας, της αποδοτικότητάς και βιωσιμότητας των μεταφορών έχει αναπτύξει ένα ITS σύστημα που περιλαμβάνει μια σειρά από διάφορες τεχνολογίες και εφαρμογές [12]. Μέσα από το σύστημα αυτό παρέχετε γνώση για την κυκλοφορία στους οδηγούς σε πραγματικό χρόνο μέσα από ηλεκτρονικές πινακίδες, ραδιοφωνικές εκπομπές και συστήματα πλοήγησης αυτοκινήτων [12]. Για την βελτίωση της ροής της κυκλο-

φορίας έχει αναπτυχθεί ένα σύστημα για την αυτόματη συλλογή της χρέωσης διοδίων στους αυτοκινητόδρομους, μειώνοντας την ανάγκη των οδηγών για στάση [12]. Παράλληλα μέσα από συστήματα διαχείρισης της κυκλοφορίας γίνεται παρακολούθηση της κατάστασης στο οδικό δίκτυο που συμβάλουν στην αντιμετώπιση της συμφόρησης και την άμεση ανταπόκριση σε αναγκαίες καταστάσεις [12].

Η Νέα Υόρκη αποτελεί ένα διαφορετικό παράδειγμα πόλης που χρησιμοποιεί Έξυπνα Συστήματα Μεταφορών (ITS) για την καταπολέμηση της κυκλοφοριακής συμφόρησης [20]. Μέσω ενός εκτεταμένου δικτύου αισθητήρων, συλλέγονται δεδομένα σε πραγματικό χρόνο και επεξεργάζονται μαζί με ιστορικά δεδομένα, προκειμένου να ληφθούν ενημερωμένες αποφάσεις για τη διαχείριση της κυκλοφορίας. Οι πληροφορίες αυτές παρέχονται στους οδηγούς μέσω ηλεκτρονικών πινακίδων κατά μήκος των εθνικών οδών και των κύριων αρτηριών, βοηθώντας τους να λάβουν ενημερωμένες αποφάσεις για την κυκλοφορία τους. Επιπλέον, μέσω ενός κέντρου ελέγχου, γίνεται ο συντονισμός των αισθητήρων με στόχο τη βελτίωση της κυκλοφορίας και την αποτελεσματική αντιμετώπιση των ατυχημάτων [5].

Στην Ελλάδα η ιδιωτική εταιρία Αττική Οδός λειτουργεί ένα μέρος του εθνικού οδικού δικτύου. Αυτό πραγματοποιείται με την χρήση ενός συστήματος διαχείρισης κίνησης που περιλαμβάνει σταθμούς εργασίας με πρόσβαση σε λογισμικό διαχείρισης κίνησης και ελέγχου του εξοπλισμού [2]. Το σύστημα αυτό συλλέγει δεδομένα από αισθητήρες που είναι διατεταγμένοι στο οδικό δίκτυο με στόχο να γίνεται η παρακολούθηση και ο έλεγχος της κίνησης. Παράλληλα παρέχει λύσεις όπως αυτή της αυτόματης πληρωμής διοδίων [2].

Κεφάλαιο C

Εργαλεία που χρησιμοποιήθηκαν

Παρακάτω παρουσιάζονται και αναλύονται τα κύρια εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής. Κάθε ένα από αυτά τα εργαλεία διαδραματίζει ζωτικό ρόλο στην ανάπτυξη και την απόδοση της. Αυτά τα εργαλεία περιλαμβάνουν την Angular, ένα δημοφιλές framework για την ανάπτυξη εφαρμογών στην πλευρά του πελάτη, το Spring Boot, ένα δημοφιλές framework που επιταχύνει την ανάπτυξη της εφαρμογής, το Maven, ένα εργαλείο διαχείρισης έργων που χρησιμοποιείται ευρέως στην ανάπτυξη λογισμικού Java, και το Docker, μια πλατφόρμα ανοιχτού κώδικα που επιτρέπει την αυτοματοποίηση της εγκατάστασης, της διαμόρφωσης και της εκτέλεσης εφαρμογών λογισμικού σε ελαφριές και φορητές δομές, γνωστές ως containers.

C.0.1 Spring Boot

Το Spring Boot είναι ένα framework ανοιχτού κώδικα το οποίο αναπτύχθηκε από την Pivotal Team. Η πρώτη έκδοση του κυκλοφόρησε τον Απρίλιο του 2004 [11] και έκτοτε έχει λάβει αρκετές αναβαθμίσεις με την μεγαλύτερη να λαμβάνει χώρα τον Νοέμβριο του 2022 με την κυκλοφορία του Spring Boot 3.

Το Spring Boot αποτελεί επέκταση του Spring Framework και κάνει την ανάπτυξη εφαρμογών και microservices ταχύτερη και ευκολότερη[13]. Κύριο χαρακτηριστικό του είναι η ευκολία που παρέχει στους προγραμματιστές στο να δημιουργούν αυτόνομες εφαρμογές που είναι έτοιμες για παραγωγή (production-ready), προσφέροντας αυτόματη ρύθμιση παραμέτρων και προκαθορισμένα πρότυπα (auto-configuration and predefined templates) [13]. Παρακάτω αναλύονται τα βασικά χαρακτηριστικά του Spring Boot.

- **Αυτόματη διαμόρφωση (Autoconfiguration)** Παρέχει αυτόματη διαμόρφωση της εφαρμογής ανάλογα με βάση τα dependencies που έχουν προστεθεί[13]. Απόρροια αυτού είναι ο προγραμματιστής να επικεντρώνεται στην ανάπτυξη της εφαρμογής και όχι στην παραμετροποίηση της.
- **Αυτόνομες εφαρμογές** Με το Spring Boot μπορούμε πολύ εύκολα να δημιουργήσουμε μια αυτόνομη εφαρμογή, προσφέροντας μας λύσεις όπως είναι οι ενσωματωμένοι web servers όπως οι Tomcat, Jetty και Undertow[21]
- **Χαρακτηριστικά για παραγωγή** Παρέχει χαρακτηριστικά όπως ο έλεγχος υγείας της εφαρμογής και μετρήσεις πόρων που συμβαδίζουν με τις ανάγκες μιας σύγχρονης εφαρμογής[21].
- **Προκαθορισμένα πρότυπα** Το Spring Boot παρέχει μια λίστα από πληθώρα dependencies που μπορούν να χρησιμοποιηθούν, οι οποίες απλοποιούν την παραμετροποίηση της εφαρμογής, επιταχύνοντας τον χρόνο ανάπτυξης της[23].

Ως επέκταση του Spring Framework, το Spring Boot φέρει αρκετά χαρακτηριστικά που αποτελούν και τον λόγο για τον οποίο αρκετοί προγραμματιστές το επιλέγουν σε σχέση με το Spring. Κάποιοι από αυτούς συσχετίζονται με τους ενσωματωμένους web server που αναφέρθηκαν και πριν ή και ακόμα και την ελάχιστη παραμετροποίηση (configuration) που απαιτεί. Πιο αναλυτικά οι διαφορές των δύο αυτών framework φαίνεται στον παρακάτω πίνακα.

Πίνακας C.1: Σύγκριση *Spring* και *Spring Boot*.

Feature	Spring	Spring Boot
Autoconfiguration	No	Yes
Standalone Applications	No	Yes
Inversion of Control (IoC)	Yes	Yes
Dependency Injection (DI)	Yes	Yes
Embedded Servers	No	Yes (Tomcat, Jetty, Undertow)

C.0.2 Maven

Το Maven είναι ένα εργαλείο διαχείρισης και κατανόησης έργων λογισμικού που χρησιμοποιείται κυρίως με έργα που βασίζονται σε Java. Αναπτύχθηκε από την Apache Software Foun-

dition με την πρώτη σταθερή έκδοση να διατίθεται στις 13 Ιουλίου του 2004 [16]. Στόχος του είναι να αυτοματοποιήσει την διαδικασία με την οποία γίνεται η λήψη και προσθήκη των βιβλιοθηκών, Jar αρχείων και εξαρτήσεων (dependencies) στο εκάστοτε έργο.

Ως προς τον τρόπο λειτουργίας του, το maven ορίζεται από ένα XML αρχείο pom.xml (Project Object Model) που περιέχει διάφορες πληροφορίες για το έργο και λεπτομέρειες για την διαμόρφωσή του. Πιο αναλυτικά τα χαρακτηριστικά του αναφέρονται στον παρακάτω πίνακα [25].

Πίνακας C.2: Χαρακτηριστικά του Maven

Χαρακτηριστικό	Περιγραφή
Δομή έργου	Δημιουργεί φακέλους για τον πυγαίο κώδικα (src/main/java) και για τον testing κώδικα της εφαρμογής (src/test/java)
Εξαρτήσεις (Dependencies)	Διαχειρίζεται αυτόματα τις βιβλιοθήκες και εξαρτήσεις που θέλουμε να προσθέσουμε στο έργο μας, κάνοντας λήψη αυτών από ένα κεντρικό αποθετήριο στο τοπικό αποθετήριο
Κύκλος ζωής και κατασκευής	Καθορίζουν μια σειρά από φάσεις και στόχους που μπορούν για παράδειγμα να κατασκευάσουν την εφαρμογή (δημιουργία Jar αρχείων) ή και να ανεβάσουν και να εκτελέσουν αυτά τα αρχεία σε κάποιο web server(deploy)
Αρχέτυπα	Παρέχει μια σειρά από πρότυπες δομές έργων με στόχο να επιταχύνει την ανάπτυξη της εφαρμογής
Πρόσθετα	Το maven χρησιμοποιεί πρόσθετα προσφέροντας επιπλέον παραμετροποίηση της εφαρμογής

C.0.3 Angular

Η Angular είναι ένα framework ανοιχτού κώδικα για την δημιουργία προηγμένων και επεκτάσιμων εφαρμογών ιστού στην πλευρά του πελάτη (client-side). Δημιουργήθηκε από την Google με την πρώτη έκδοση της να κυκλοφορεί στις αρχές Σεπτεμβρίου 2016. Ως γλώσσα προγραμματισμού χρησιμοποιεί την TypeScript με στόχο να μειώσει τον περιττό κώδικα, ώστε να δημιουργούμε πιο γρήγορες και πιο “ελαφριές” εφαρμογές με γνώμονα την απόδοση.

Ως προς τον τρόπο λειτουργίας, η Angular ακολουθεί μια αρχιτεκτονική από στοιχεία και υπηρεσίες γνωστά και ως components και services. Το κάθε στοιχείο είναι αυτόνομο και περιέχει τον δικό του HTML, CSS και TypeScript κώδικα. Κάθε ένα από αυτά είναι υπεύθυνο για [26]

- **HTML** για το τι προβάλετε στην οθόνη του χρήστη
- **CSS** για την μορφοποίηση του περιεχομένου που προβάλετε
- **TypeScript** για τον καθορισμό της συμπεριφοράς του εκάστοτε component

Όσο αφορά τα services, αποτελούνται από TypeScript κλάσεις οι οποίες μας βοηθάνε να διαχωρίσουμε τα δεδομένα από τις λειτουργίες της Angular εφαρμογής μας. Οι λειτουργίες που ορίζουμε ότι θα έχει κάποιο service μπορούν να χρησιμοποιηθούν από πολλαπλά components, μειώνοντας έτσι τον διπλότυπο κώδικα. Κάποιες άλλες λειτουργίες που μπορούν να δημιουργηθούν στα services είναι η λήψη και η διαχείριση των δεδομένων. Πιο αναλυτικά μπορούμε μέσα από τα services να λαμβάνουμε δεδομένα που θέλουμε να χρησιμοποιήσει η εφαρμογή μας και να τα διανείμουμε στα διάφορα components που τα χρειάζονται κάνοντας των κώδικα της εφαρμογής επαναχρησιμοποιούμενο.

C.0.4 Docker

Το Docker αποτελεί ένα λογισμικό ανοιχτού κώδικα που μας επιτρέπει να αναπτύσσουμε λογισμικό σε απομονωμένα Containers. Ο κύριος στόχος του [3] είναι οποιαδήποτε εφαρμογή να μπορέσει να αναπτυχθεί και να τρέξει σε οποιοδήποτε εξυπηρετητή οπουδήποτε. Η ιδέα για αυτό το λογισμικό ξεκίνησε στην Silicon Valley και αμέσως τράβηξε την προσοχή μεγάλων κολοσσών όπως η Google, Amazon, Microsoft, Red Hat οι οποίοι υποστηρίζουν το Docker στις πλατφόρμες τους, αυξάνοντας έτσι την εικόνα του λογισμικού και της ευχρηστίας του. Σε σχέση με τις υπόλοιπες τεχνολογίες εικονοποίησης, το Docker δημιουργεί ένα προτυποποιημένο περιβάλλον για την ανάπτυξη εφαρμογών. Αφού δημιουργήσουμε την εικόνα της εφαρμογής μας μπορούμε να την τρέξουμε σε οποιοσδήποτε σύστημα που έχει εγκατεστημένο το Docker Engine. Απόρροια αυτού είναι ο δημιουργός της εφαρμογής να εξοικονομεί χρόνο από το να προσαρμόσει την εφαρμογή του ώστε να υποστηρίζεται από πολλές πλατφόρμες και λειτουργικά συστήματα. Το γεγονός ότι η εφαρμογή οργανώνεται σε αυτόνομα και απομονωμένα containers λύνει αρκετά προβλήματα. Ένα από αυτά αποτελεί την γρήγορη αντικατάσταση κάποιου container σε περίπτωση που οι ανάγκες της εφαρμογής το απαιτούν.

Επιπλέον αξίζει να σημειωθεί ότι το Docker μπορεί να συνδυαστεί με επιπλέον εργαλεία που ενισχύουν τις δυνατότητες τους. Ένα από αυτά είναι το Kubernetes, που παρουσιάζουν για πρώτη φορά την ιδέα των pods και της ενορχήστρωσης των containers. Πιο αναλυτικά με το Kubernetes χωρίζουμε την εφαρμογή σε ομάδες από containers τα οποία είναι στενά συνδεδεμένα μεταξύ τους και αντιμετωπίζονται ως η μικρότερη μονάδα προς ανάπτυξη. Ο συνδυασμός αυτών των εργαλείων θα μπορούσε να βρεθεί σε ποιο σύνθετα και πολύπλοκα συστήματα.

C.1 Παρουσίαση εφαρμογής

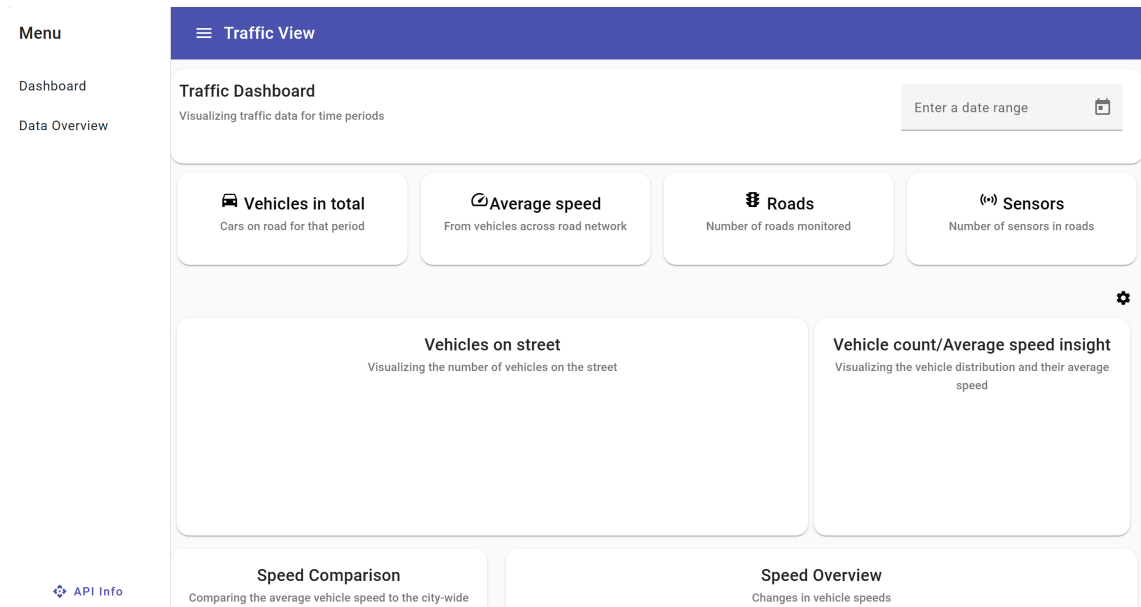
Στην παρακάτω ενότητα, θα εξετάσουμε λεπτομερώς την εφαρμογή που αναπτύχθηκε στα πλαίσια της εργασίας. Θα αναλύσουμε τις βασικές λειτουργίες της, τον τρόπο χρήσης της και τον τρόπο με τον οποίο συμβάλλει στην κατανόηση και την ανάλυση της κυκλοφορίας στο οδικό δίκτυο. Στόχος ήταν η ανάπτυξη μιας διαδραστικής εφαρμογής μέσα από την οποία ο χρήστης μπορεί να πλοηγηθεί σε διάφορα τμήματα της καθώς και η εφαρμογή να προσφέρει και την απαραίτητη ανάδραση στον χρήστη ώστε να ενισχύσουμε την εμπειρία χρήσης της. Πιο αναλυτικά η εφαρμογή αποτελείται από τρία βασικά τμήματα. Το μενού που μέσα από αυτό ο χρήστης μπορεί να πλοηγηθεί στην εφαρμογή και να εξερευνήσει τις διάφορες λειτουργίες της, το toolbar που περιέχει το όνομα της εφαρμογής και ένα κουμπί που μας δίνει την επιλογή να ανοίξουμε και να κλείσουμε το μενού και τέλος το κύριο περιεχόμενο της εφαρμογής.

C.1.1 Dashboard

Με το που εισέρθει ο χρήστης στην εφαρμογή θα ανακατευθυνθεί στην ενότητα του dashboard. Τα dashboards αποτελούν εργαλεία που χρησιμοποιούνται για την παρακολούθηση, την ανάλυση και την εμφάνιση δεδομένων με έναν τρόπο φιλικό προς τον χρήστη. Μια εύλογη σύγκριση που θα μπορούσαμε να κάνουμε για να αναλύσουμε την έννοια και τον ρόλο των dashboards θα ήταν αυτή με το ταμπλό ενός αυτοκινήτου. Όπως το ταμπλό λοιπόν έτσι και τα dashboards εμφανίζουν σημαντικές πληροφορίες με μια ματιά για να βοηθήσουν τον χρήστη να κατανοήσει σε σύντομο χρόνο πολύτιμα και κρίσιμα δεδομένα.

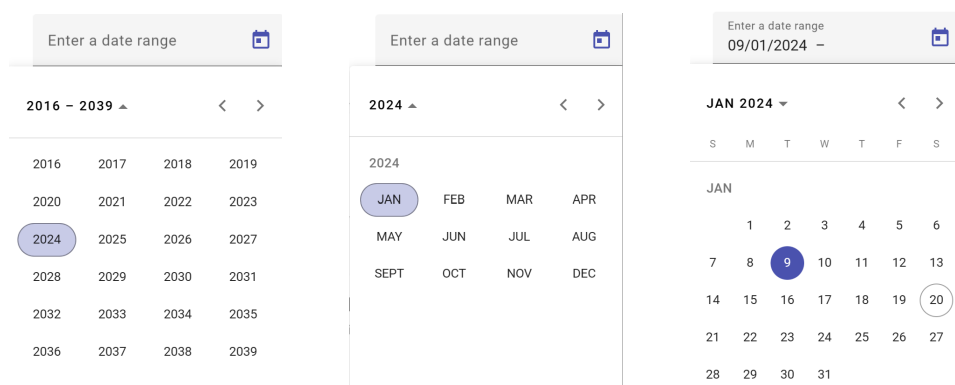
Σε συνέχεια της ανάλυσης της εφαρμογής αφού ο χρήστης εισέρθει στην εφαρμογή μπορεί να δει το dashboard. Στόχος του dashboard είναι να οπτικοποιεί δεδομένα από την κατάσταση στο οδικό δίκτυο για τις χρονικές περιόδους που επιλέξει ο χρήστης καθώς και να του δίνεται

η δυνατότητα να προσαρμόσει την πληροφορία που θέλει να δει μέσα από διάφορα φίλτρα (Σχήμα C.1).



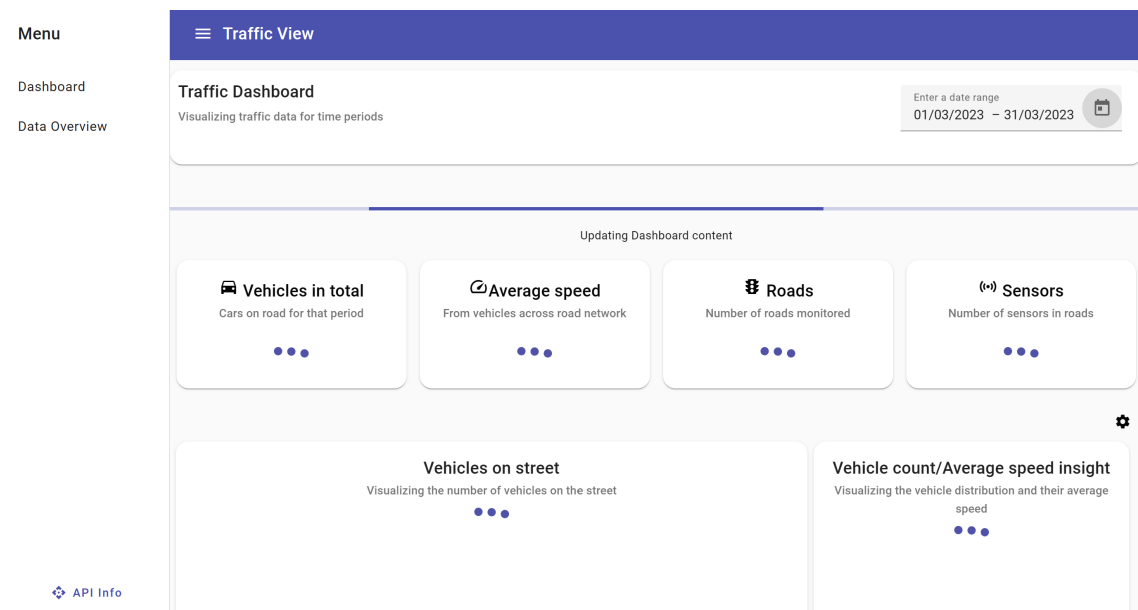
Σχήμα C.1: Αρχική οθόνη εφαρμογής

Η επιλογή της χρονικής περιόδου γίνεται μέσα από ένα διαδραστικό ημερολόγιο που υπάρχει πάνω δεξιά (Σχήμα C.1). Μέσα από αυτό το ημερολόγιο ο χρήστης μπορεί να επιλέξει το έτος, τον μήνα ή μήνες καθώς και την μέρα ή μέρες για τις οποίες επιθυμεί να ανακτήσει και να προβληθεί η πληροφορία (Σχήμα C.2).



Σχήμα C.2: Ημερολόγιο για την επιλογή της χρονικής περιόδου

Έχοντας επιλέξει ο χρήστης την χρονική περίοδο που θα ήθελε να δει τις οπτικοποιήσεις από την κατάσταση του οδικού δικτύου η εφαρμογή θα ξεκινήσει να συλλέγει και να επεξεργάζεται τα δεδομένα με στόχο να εξάγουμε σημαντικές πληροφορίες από αυτά. Στο διάστημα αυτό διάφορα animation εμφανίζονται στην οθόνη του χρήστη δείχνοντας του ότι η εφαρμογή έχει λάβει το αίτημα του και δημιουργεί το dashboard με τα δεδομένα. Τα animation αυτά περιλαμβάνουν την μπάρα που αναγράφει το κείμενο “Updating Dashboard content” και τις φουσαλίδες που βρίσκονται σε κάθε panel που κινούνται σε κυματική μορφή (Σχήμα C.3).



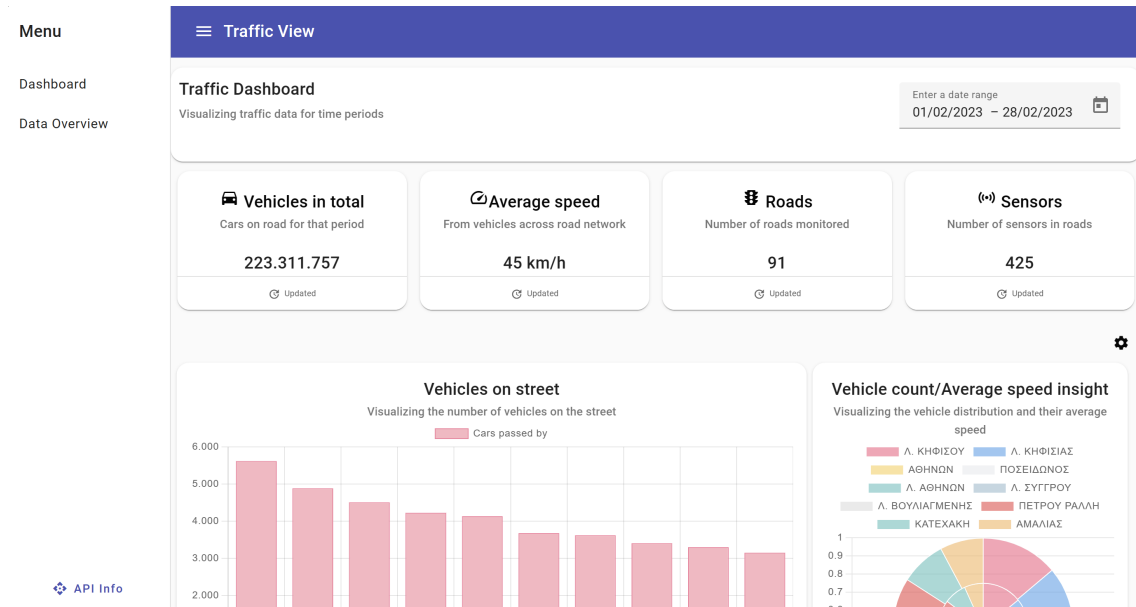
Σχήμα C.3: Animation αφού έχει επιλεχθεί η χρονική περίοδος

Μόλις ολοκληρωθεί η διαδικασία αυτή διάφορα διαγράμματα και κάποια σχετική πληροφορία θα εμφανιστούν στην οθόνη. Σαν ενδεικτικό παράδειγμα θα χρησιμοποιήσουμε την χρονική περίοδο του Φεβρουαρίου του 2023 για να μελετήσουμε τις οπτικοποιήσεις.

Τα τέσσερα πρώτα panel που είναι διατεταγμένα στην σειρά μας δείχνουν μια συνολική εικόνα για το πως ήταν η κίνηση στο οδικό δίκτυο (Σχήμα C.4). Συγκεκριμένα περιλαμβάνει τον συνολικό αριθμό οχημάτων που καταγράφηκε, την μέση ταχύτητα όλων των οχημάτων από όλο το δίκτυο, τον συνολικό αριθμό από δρόμους για τους οποίους έχουμε καταγραφές καθώς και τον αριθμό των αισθητήρων που υπάρχουν σε αυτούς.

Με μια γρήγορη ματιά ο χρήστης μπορεί να διατυπώσει πως για την περίοδο του Φεβρουαρίου του 2023 έχουν καταγραφεί 223.311.757 οχήματα με συνολική μέση ταχύτητα τα 45 χιλιόμετρα

ανά ώρα. Οι καταγραφές αυτές προκύπτουν από 91 διαφορετικούς δρόμους της Αθήνας και έχουν συλλεχθεί από 425 αισθητήρες που υπήρχαν διαθέσιμη για εκείνη την ημέρα

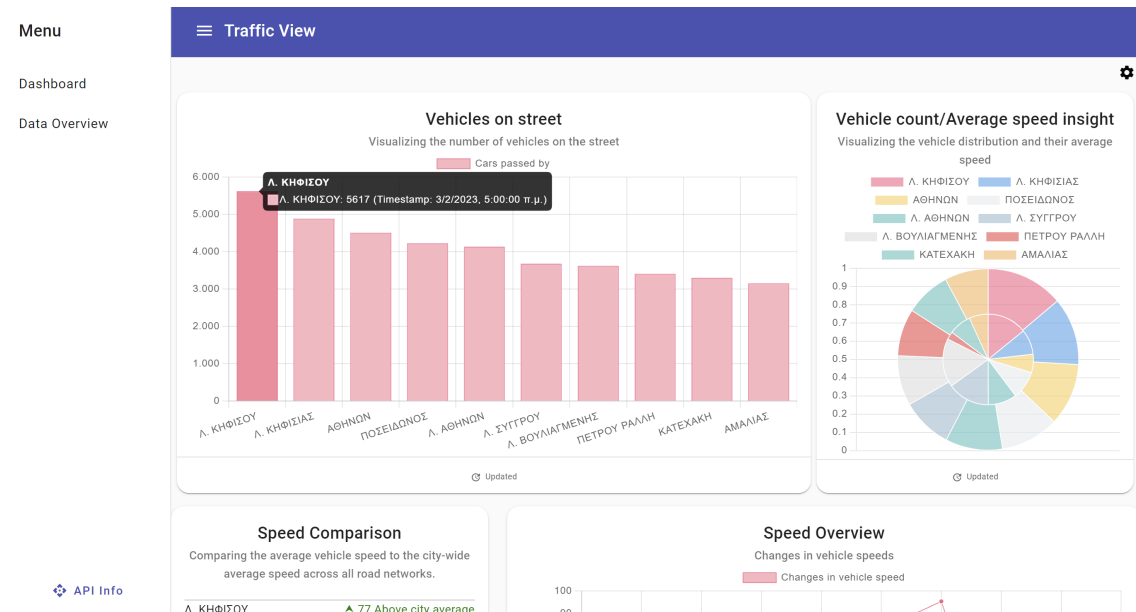


Σχήμα C.4: Dashboard μετά την επιλογή ημερομηνίας για την οπτικοποίηση δεδομένων

Στο πιο κάτω τμήμα του dashboard παρουσιάζονται πιο αναλυτικά τα δεδομένα. Προκαθορισμένα ο χρήστης θα βλέπει στην οθόνη του δεδομένα που αφορούν τους δρόμους με την περισσότερη κίνηση, αυτό υπολογίζεται με βάση τον αριθμό των οχημάτων που πέρασαν από τον συγκεκριμένο δρόμο για την χρονική περίοδο που έχει επιλεχθεί. Η ανάλυση που γίνεται από το dashboard αφορά

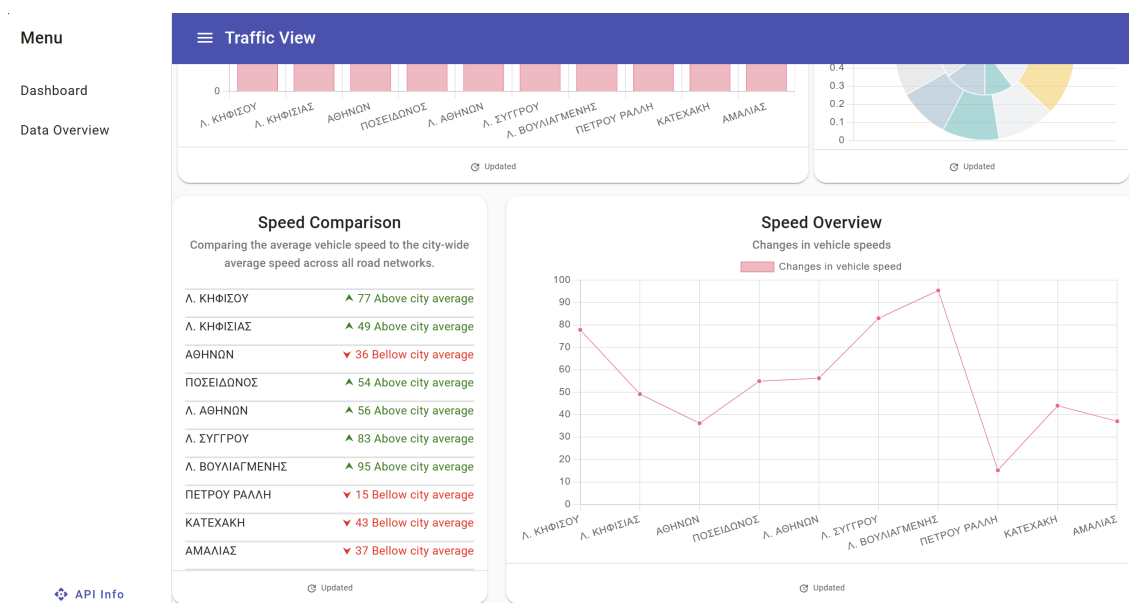
1. Ένα ραβδόγραμμα που μας παρουσιάζει τους δρόμους με την μεγαλύτερη διέλευση οχημάτων (Σχήμα C.5).
2. Ένα διάγραμμα πίτας που μας παρουσιάζει συνδυαστικά τον αριθμό των οχημάτων και την μέση ταχύτητα τους για εκείνο τον δρόμο, ώστε να έχουμε μια ποσοστιαία εικόνα (Σχήμα C.5)
3. Ένα πίνακα που μας δείχνει τις κατατάξεις από τους δρόμους αυτούς όσο αφορά την μέση ταχύτητα των οχημάτων για εκείνη την περίοδο και αν βρίσκεται πάνω, κάτω ή ακόμα και αν ισούται με το γενικό μέσο όρο της πόλης (Σχήμα C.6).

4. Μια γραφική παράσταση γραμμής που αναπαριστά την ταχύτητα στους δρόμους αυτούς (Σχήμα C.6)



Σχήμα C.5: Αναπαράσταση δεδομένων κυκλοφορίας σχετικά με τον αριθμό οχημάτων

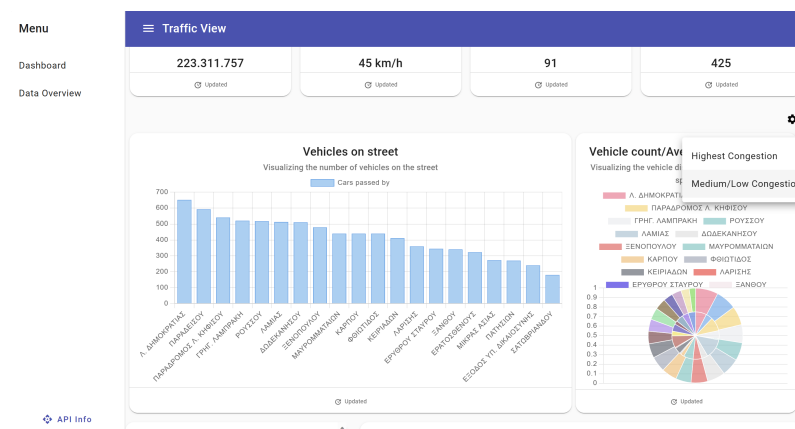
Στην παρακάτω εικόνα βλέπουμε τις οπτικοποιήσεις που περιγράψαμε στο 3. και 4. Σχετικά με τις συγκρίσεις της μέσης ταχύτητας κάθε δρόμου με τον γενικό μέσο όρο ταχύτητας ολόκληρης της πόλης και η αναπαράστασή τους σε μια γραφική παράσταση γραμμής (Σχήμα C.6). Είναι σημαντικό να σημειώσουμε ότι κατά την κίνηση του δείκτη πάνω από τα διαγράμματα, ο χρήστης έχει τη δυνατότητα να εμφανίσει πληροφορίες pop-up, όπως αυτή που παρουσιάζεται στο Σχήμα C.5. Στο συγκεκριμένο παράδειγμα βλέπουμε ότι για τον μήνα Φεβρουάριο ο μεγαλύτερος αριθμός οχημάτων καταγράφηκε στην Λεωφόρο Κηφισού με συνολικό αριθμό οχημάτων 5617 που πραγματοποιήθηκε στις 3 Φεβρουαρίου. Αντίστοιχη λογική ακολουθούν όλα τα διαγράμματα που υπάρχουν στην εφαρμογή.



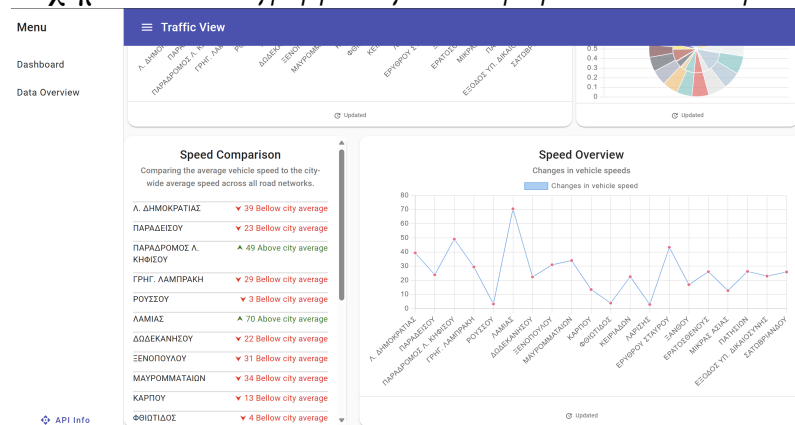
Σχήμα C.6: Αναπαράσταση δεδομένων κυκλοφορίας σχετικά με την μέση ταχύτητα των οχημάτων

Αντίστοιχα μέσω της εφαρμογής προσφέρεται και οπτικοποίηση μέτριας έως και λίγης κίνησης στον δρόμο (Σχήμα C.9). Για να αποκτήσει πρόσβαση σε αυτήν την πληροφορία, ο χρήστης πρέπει να πατήσει το εικονίδιο του γραναζιού που βρίσκεται πάνω από τον πίνακα ελέγχου. Αυτό θα ανοίξει ένα αναπτυσσόμενο μενού με δύο επιλογές: 'Highest Congestion' ή 'Medium/Low Congestion'. Επιλέγοντας την επιλογή 'Medium/Low Congestion', το περιεχόμενο του dashboard ανανεώνεται αυτόματα για να παρουσιάσει τα αντίστοιχα δεδομένα. Οι οπτικοποιήσεις τους εμφανίζονται ταξινομημένες στα διαγράμματα με σκοπό την ευκολότερη ανάγνωση και κατανόηση τους (Σχήμα C.9).

Μια σημαντική λεπτομέρεια είναι ότι τα χρώματα των διαγραμμάτων αλλάζουν ανάλογα με τα δεδομένα που εμφανίζονται από Highest Congestion σε Low/Medium Congestion. Αυτό παρέχει ένα οπτικό ερέθισμα στον χρήστη, επιτρέποντάς του να κατανοήσει την αλλαγή των δεδομένων με μια ματιά. Αυτό συμβάλλει στη βελτιστοποίηση της επικοινωνίας των πληροφοριών και καθιστά την εφαρμογή πιο διαδραστική και ευχάριστη στη χρήση.



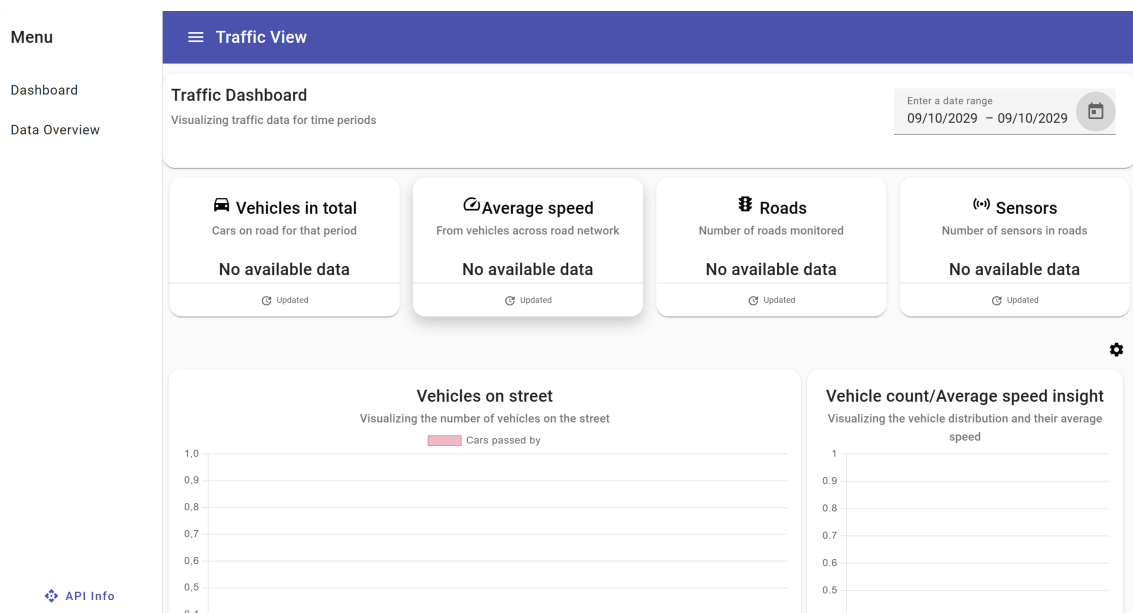
Σχήμα C.7: Διαγράμματα για τον αριθμό των αυτοκινήτων



Σχήμα C.8: Διαγράμματα για την ταχύτητα των οχημάτων

Σχήμα C.9: Dashboard για τους δρόμους με μέτρια έως και λίγη κίνηση

Στην περίπτωση κατά την οποία ο χρήστης επιλέξει μια χρονική περίοδο για την οποία δεν υπάρχουν διαθέσιμα δεδομένα, η εφαρμογή θα παρέχει ενημέρωση μέσω των αντίστοιχων μηνυμάτων. Πιο συγκεκριμένα στα πάνελ επάνω, θα εμφανίζεται το μήνυμα 'No Available Data', επισημαίνοντας έτσι την απουσία διαθέσιμων δεδομένων για τη συγκεκριμένη χρονική περίοδο (Σχήμα C.10).



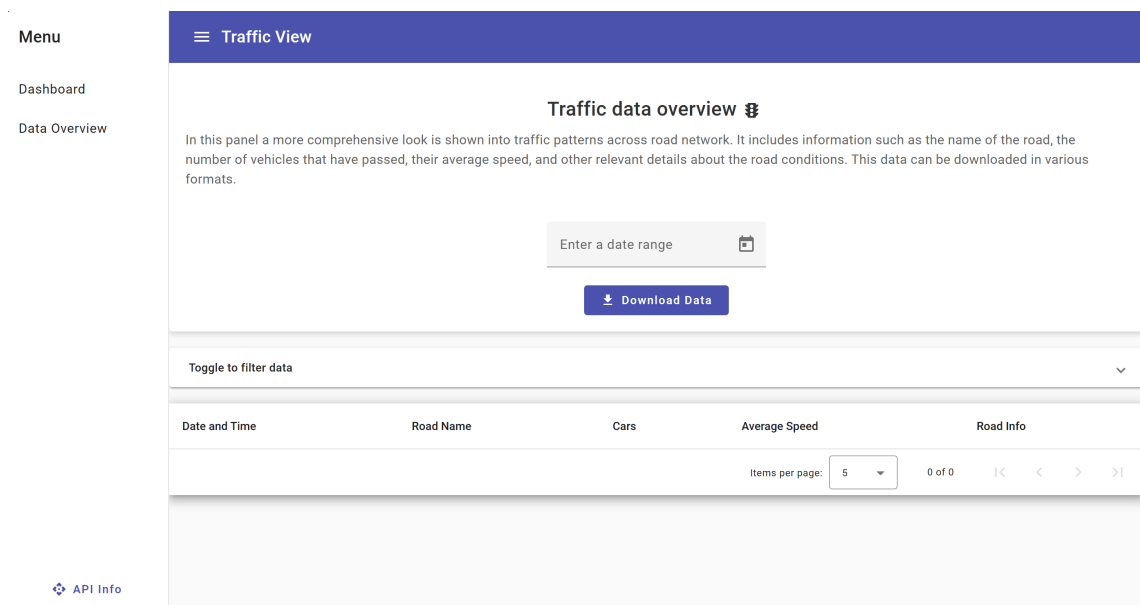
Σχήμα C.10: Μήνυμα μη διαθέσιμων δεδομένων από την εφαρμογή

Μετά την ανάλυση των δεδομένων που παρουσιάζονται στα dashboard, είμαστε σε θέση να μελετήσουμε τα δεδομένα που παρέχονται και να αποκτήσουμε πολύτιμες γνώσεις σχετικά με τα μοτίβα κυκλοφορίας στο οδικό δίκτυο. Αναφερόμενοι στο παράδειγμα που εξετάσαμε προηγουμένως, για τον μήνα Φεβρουάριο του 2023, μπορούμε να εξάγουμε πλέον χρήσιμες πληροφορίες που αφορούν την κυκλοφορία στο οδικό δίκτυο της Αθήνας.

Για την συγκεκριμένη χρονική περίοδο παρατηρούμε πως η Λεωφόρος Κηφισού ήταν αυτή η οποία συγκέντρωσε τον μεγαλύτερο αριθμό οχημάτων. Η καταγραφή αυτή έγινε στις 3 Φεβρουαρίου και τα δεδομένα αντλήθηκαν μέχρι και 5 π.μ. Κατά μέσο όρο τα οχήματα που κινούνταν κατά μήκος της λεωφόρου είχαν μέση ταχύτητα τα 77 χιλιόμετρα ανά ώρα τα οποία ξεπερνούν τον μέσο όρο ταχύτητας ολόκληρης της πόλης που για τον μήνα Φεβρουάριο έχει υπολογιστεί στα 45 χιλιόμετρα ανα ώρα. Αντίστοιχα παρατηρούμε ότι η Λεωφόρος Βουλιαγμένης παρά τον αυξημένο αριθμό διελικομένων οχημάτων, τα οχήματα που κινούνταν κατά μήκος της διατήρησαν υψηλότερο μέσο όρο ταχύτητας. Αυτό μπορεί να υποδεικνύει ότι, πιθανόν, οι οδηγοί προτιμούν να κινούνται με μεγαλύτερη ταχύτητα στη Λεωφόρο Βουλιαγμένης, ίσως λόγω καλύτερων οδικών συνθηκών ή λιγότερης κυκλοφοριακής συμφόρησης, επιτρέποντάς μας να κατανοήσουμε τα πρότυπα και τις τάσεις που επηρεάζουν την κυκλοφορία

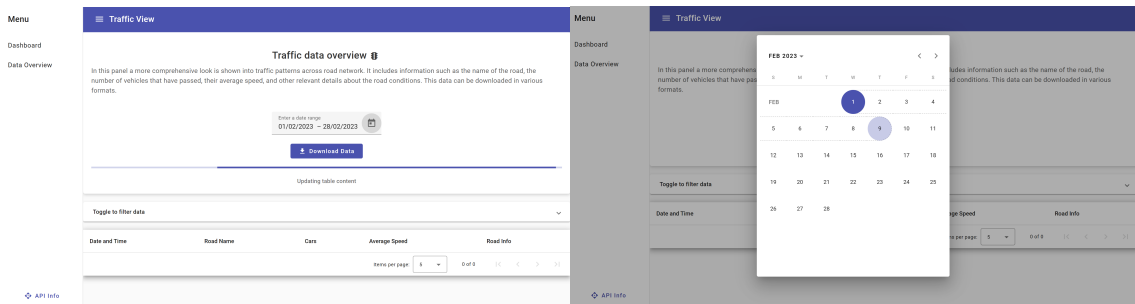
C.1.2 Data Overview

Αν και τα dashboard μπορούν σε ελάχιστον χρονικό διάστημα να μας παρουσιάσουν κρίσιμες πληροφορίες με φιλικό τρόπο, πολλές φορές προκύπτει η ανάγκη για περεταίρω μελέτη των δεδομένων και η χρήση τους σε εξωτερικά εργαλεία. Την ανάγκη αυτή καλύπτει μια διαφορετική ενότητα της εφαρμογής που ονομάζεται Data Overview (Σχήμα C.11). Ο χρήστης μπορεί να επιλέξει να εισέρθει στην ενότητα αυτή μέσω του μενού που υπάρχει στο πλάι. Σαν τρόπο χρήσης της ακολουθείτε παρόμοιο μοτίβο με το dashboard, ο χρήστης καλείται να επιλέξει μια χρονική περίοδο, τα δεδομένα εμφανίζονται στην οθόνη καθώς και του δίνεται η δυνατότητα να φιλτράρει τα δεδομένα αυτά.



Σχήμα C.11: Παρουσίαση ενότητας Data Overview

Όταν ο χρήστης πατήσει για να επιλέξει την χρονική περίοδο για την οποία θέλει να δει αναλυτικά τα δεδομένα που υπάρχουν διαθέσιμα, ένα αναδυόμενο ημερολόγιο θα εμφανιστεί στην οθόνη του. Η λειτουργία του ημερολογίου αυτή είναι η ίδια με αυτή που περιεγράφηκε στην προηγούμενη ενότητα, δηλαδή ο χρήστης μπορεί να επιλέξει το έτος, μήνα και μέρες για τις όποιες θέλει να αντλήσει τα δεδομένα του (Σχήμα C.13). Αμέσως μετά τα αντίστοιχα animation θα εμφανιστούν στην οθόνη του χρήστη δείχνοντας τους ότι η εφαρμογή αντλεί τα δεδομένα τα οποία θα παρουσιάσει (Σχήμα C.12)



Σχήμα C.12: Animation εφαρμογής

Σχήμα C.13: Αναδυόμενο ημερολόγιο

Σχήμα C.14: Data Overview επιλογή χρονικής περιόδου και ανάκτηση δεδομένων

Αφού η εφαρμογή έχει συλλέξει όλα τα διαθέσιμα δεδομένα, θα τα παρουσιάσει σε μια μορφή δυναμικού πίνακα όπως φαίνεται παρακάτω (Σχέδιο C.15). Μέσα από αυτόν τον πίνακα ο χρήστης μπορεί να δει αναλυτικά τις παρακάτω πληροφορίες

- **Date and Time:** Αποτελεί την χρονική στιγμή που έγινε η καταγραφή.
- **Road Name:** Παρουσιάζει το όνομα του δρόμου.
- **Cars:** Αποτελεί τον αριθμό των οχημάτων που καταγράφηκαν στον δρόμο.
- **Average Speed:** Αποτελεί την μέση ταχύτητα στον δρόμο αυτό.
- **Road Info:** Παρουσιάζει κάποια σχετική πληροφορία για την κατεύθυνση του δρόμου.

Επιπλέον μέσα από αυτόν τον πίνακα ο χρήστης μπορεί να ελέγξει το περιεχόμενο που θα προβάλλεται καθώς και να πλοηγηθεί σε αυτόν. Παράλληλα δίνεται η δυνατότητα στο να φιλτράρουμε τα δεδομένα με βάση συγκεκριμένους δρόμους, μέρες, και μέση ταχύτητα. Για να έχουμε πρόσβαση στην λειτουργία αυτή ο χρήστης θα πρέπει να ανοίξει το επεκτάσιμο πάνελ που αναγράφει Toggle to filter data και εκεί θα έχει την επιλογή να πληκτρολογήσει και να περιορίσει τα δεδομένα που βλέπει στον πίνακα με τα επιθυμητά (Σχήμα C.15).

Επεκτείνοντας το παράδειγμα που μελετήσαμε πριν για τον μήνα Φεβρουάριο βλέπουμε τα αναλυτικά δεδομένα που ανέκτησε η εφαρμογή μας και ο αριθμός τους ανέρχεται στις 238.562 καταγραφές. Κάτω από τον πίνακα φαίνεται η επιλογή που δίνει στον χρήστη να προσαρμόσει το περιεχόμενο του. Δηλαδή πόσες εγγραφές να εμφανίζονται σε μια καρτέλα του πίνακα. Μέσα από τα βελόνια που εμφανίζονται ακριβώς από δίπλα ο χρήστης μπορεί να πλοηγηθεί στις καρτέλες αυτές. Οι επιλογές για το πόσο περιεχόμενο θα εμφανίζεται στον πίνακα είναι

5 εγγραφές, 10, 25, 50 και 100. Επιπλέον πατώντας σε έναν οποιοδήποτε τίτλο (Date and Time, Road Name, Cars, Average Speed, Road Info) του πίνακα μπορεί να ταξινομήσει το περιεχόμενο του αλφαβητικά με αύξουσα ή φθίνουσα σειρά όπως φαίνεται στο Σχήμα C.15 πάνω από το Road Name

Menu

- Dashboard
- Data Overview

Traffic View

Traffic data overview

In this panel a more comprehensive look is shown into traffic patterns across road network. It includes information such as the name of the road, the number of vehicles that have passed, their average speed, and other relevant details about the road conditions. This data can be downloaded in various formats.

Enter a date range
01/02/2023 – 28/02/2023

Download Data

Toggle to filter data

Filter table content

Date and Time	Road Name ↑	Cars	Average Speed	Road Info
2023-02-01T00:00:00.000+00:00	Π. ΡΑΛΛΗ	381	18.55	ΚΥΡΙΟΣ ΔΡΟΜΟΣ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΑΘΗΝΑ 25 Μ. ΠΡΙΝ ΑΠΟ ΤΗ Α. ΚΗΦΙΣΟΥ (ΜΟΝΟ ΓΙΑ Φ/Σ)
2023-02-01T00:00:00.000+00:00	Λ. ΒΟΥΛΙΑΓΜΕΝΗΣ	468	78.02631578947368	ΚΥΡΙΟΣ ΔΡΟΜΟΣ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΠΡΟΣ ΚΕΝΤΡΟ, 110 Μ. ΠΡΙΝ ΑΠΟ ΤΗΝ ΠΡΙΓΚΗΠΟΣ ΠΙΕΤΡΟΥ
2023-02-01T00:00:00.000+00:00	ΚΥΠΡΟΥ	30	56	ΚΥΡΙΑ ΟΔΟΣ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΠΡΟΣ Α. ΑΘΗΝΩΝ (Β), 80Μ ΠΡΙΝ ΤΗ Α. ΑΘΗΝΩΝ
2023-02-01T00:00:00.000+00:00	ΑΘΗΝΩΝ	715	59.75	ΚΥΡΙΟΣ ΔΡΟΜΟΣ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΑΘΗΝΑ 150 Μ. ΠΡΙΝ ΑΠΟ ΤΗ ΦΑΒΙΕΡΟΥ
2023-02-01T00:00:00.000+00:00	ΑΘΗΝΩΝ	994	71.875	ΚΥΡΙΟΣ ΔΡΟΜΟΣ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΔΑΦΝΙ 150 Μ. ΠΡΙΝ ΑΠΟ ΤΗΝ ΕΥΡΥΤΑΝΙΑΣ

Items per page: 5 6 – 10 of 338562 |< < > >|

API Info

Σχήμα C.15: Εμφάνιση δεδομένων εφαρμογής και παρουσίαση δυνατότητας φιλτραρίσματος και ταξινόμησης τους

Παρακάτω εμφανίζεται ένα παράδειγμα για το τι θα μας εμφανιστεί στην περίπτωση που ο χρήστης επιλέξει να φιλτράρει το περιεχόμενο του πίνακα με βάση το όνομα του δρόμου, ώστε να του εμφανίζονται μόνο σχετικές πληροφορίες με την επιλογή του. Ο δεδομένα που φαίνονται στο Σχήμα C.16 αφορούν την περιοχή Κατεχάκη.

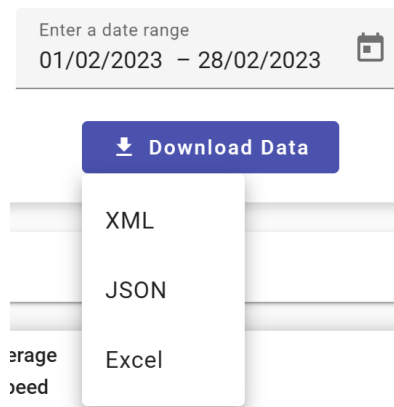
The screenshot shows a web application interface for traffic data. On the left is a menu with 'Dashboard' and 'Data Overview'. The main area is titled 'Traffic View' and 'Traffic data overview'. It includes a date range selector (01/02/2023 - 28/02/2023) and a 'Download Data' button. Below this is a 'Toggle to filter data' section with a search bar containing 'ΚΑΤΕΧΑΚΗ'. A table displays the filtered data with columns: Date and Time, Road Name, Cars, Average Speed, and Road Info. The table contains 5 rows of data for the road KATEXAKHI. At the bottom right, there is a pagination control showing 'Items per page: 5' and '1 - 5 of 11356'.

Date and Time	Road Name	Cars	Average Speed	Road Info
2023-02-01T00:00:00.000+00:00	KATEXAKHI	327	59.46341463414634	ΚΥΡΙΟΣ ΔΡΟΜΟΣ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΠΡΟΣ ΜΕΣΟΓΕΙΟΝ, 150 Μ. ΠΡΙΝ ΤΗΝ ΕΙΣΟΔΟ ΤΟΥ Γ.Ν.Α.
2023-02-01T00:00:00.000+00:00	KATEXAKHI	235	66.97560975609755	ΚΥΡΙΟΣ ΔΡΟΜΟΣ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΠΡΟΣ ΚΑΡΕΑ, 150 Μ. ΠΡΙΝ ΤΗΝ ΕΙΣΟΔΟ ΠΡΟΣ ΣΤΡΑΤΟΠΕΔΟ ΒΑΡΥΤΗ
2023-02-01T00:00:00.000+00:00	KATEXAKHI	335	59.65853658536585	ΚΥΡΙΟΣ ΔΡΟΜΟΣ ΜΕ ΚΑΤΕΥΘΥΝΣΗ Α. ΜΕΣΟΓΕΙΟΝ ΑΜΕΣΩΣ ΜΕΤΑ ΤΟΝ ΚΟΜΒΟ ΚΑΤΕΧΑΚΗ - ΣΤΡΑΤΟΠΕΔΟ ΒΑΡΥΤΗ
2023-02-01T00:00:00.000+00:00	KATEXAKHI	251	57.146341463414636	ΚΥΡΙΟΣ ΔΡΟΜΟΣ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΠΡΟΣ ΚΑΡΕΑ, 80 Μ. ΠΡΙΝ ΤΗΝ ΕΙΣΟΔΟ ΤΟΥ Γ.Ν.Α.
2023-02-01T00:00:00.000+00:00	KATEXAKHI	228	59.51219512195122	ΚΥΡΙΟΣ ΔΡΟΜΟΣ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΠΡΟΣ ΚΑΡΕΑ, 150 Μ. ΠΡΙΝ ΤΗΝ ΚΟΚΚΙΝΟΠΟΥΛΟΥ

Σχήμα C.16: Εμφάνιση φιλτραρισμένων δεδομένων για την περιοχή Κατεχάκη

Όπως αναφέραμε και στην αρχή της ενότητας, πολλές φορές προκύπτει η ανάγκη να μελετήσουμε περαιτέρω τα δεδομένα μας και αρκετές φορές η χρήση διάφορων εξωτερικών εργαλείων κρίνεται απαραίτητη. Ως λύση στην ανάγκη αυτή μέσω της εφαρμογής προσφέρεται η δυνατότητα στον χρήστη να δημιουργήσει και να κατεβάσει σε αρχεία τα δεδομένα του, ώστε πλέον να κάνουμε λόγο για διαθέσιμα δεδομένα. Οι μορφές που υποστηρίζονται για την δημιουργία αρχείων αποτελούν Excel, Json και XML οι οποίες καλύπτουν ένα αρκετά μεγάλο φάσμα αρχείων εισόδου σε μια εφαρμογή. Η λήψη των δεδομένων γίνεται μέσα από ένα drop-down μενού που εμφανίζεται όταν ο χρήστης πατήσει το κουμπί Download Data (Σχήμα C.17), στο σημείο αυτό καλείται να επιλέξει σε τι μορφή επιθυμεί να κατεβάσει τα δεδομένα του. Προϋπόθεση είναι να έχει επιλεγεί μια χρονική περίοδος ώστε να δημιουργήσουμε τα αρχεία. Στην περίπτωση που ο χρήστης δεν επιλέξει κάποια χρονική περίοδο τα αρχεία που θα κατέβουν θα περιέχουν μόνο τις απαραίτητες πληροφορίες για να θεωρούνται έγκυρα. Πιο αναλυτικά το Excel αρχείο θα περιέχει μόνο τους τίτλους από τον πίνακα, το xml θα περιέχει μόνο το xml declaration και το Json θα περιέχει τις κενές αγκύλες. Όταν ο χρήστης επιλέξει να κατεβάσει τα δεδομένα του σε αρχείο θα εμφανιστούν τα αντίστοιχα animation όπως και προηγουμένως (Σχήμα C.11)

Σημαντικό επίσης είναι να αναφέρουμε ότι τα αρχεία να ακολουθούν μια δομή που να είναι



Σχήμα C.17: Μένου για την λήψη των δεδομένων σε αρχεία

εύκολα αναγνώσιμη από τον χρήστη. Για τον λόγο αυτό γίνεται μια επιπλέον μορφοποίηση των αρχείων πριν τα λάβει ο χρήστης (Σχήμα C.18). Η μεγαλύτερη μορφοποίηση γίνεται για το αρχείο του Excel. Στην διαδικασία αυτή όχι μόνο γράφουμε τα δεδομένα μας σε ένα xlsx αρχείο, αλλά ακολουθούνται και οι παρακάτω διαδικασίες

1. Αυτόματη στοίχιση κελιών, ώστε να φαίνεται ολόκληρο το περιεχόμενο .
2. Προσθήκη μαύρων πλαισίων γύρω από το κελί, ώστε τα δεδομένα να ξεχωρίζουν.
3. Προσθήκη bold τίτλων πίνακα με αυξημένη γραμματοσειρά.

Timestamp	Sensor ID	Road Name	Counted Cars	Average Speed	Road Info
Wed Feb 01 02:00:00 EET 2023	MS863	Α. ΚΗΦΙΣΟΥ	160	26,3	ΡΑΜΠΑ ΕΙΣΟΔΟΥ ΠΡΟΣ ΙΕΡΑ ΟΔΟ ΤΟΥ ΚΛΑΔΟΥ ΤΗΣ Α. ΚΗΦΙΣΟΥ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΔΑΜΙΑ 40 Μ. ΠΡΙΝ ΑΠΟ ΤΗ
Wed Feb 01 02:00:00 EET 2023	MS878	Α. ΚΗΦΙΣΟΥ	60	39,425	ΡΑΜΠΑ ΕΙΣΟΔΟΥ ΠΡΟΣ ΚΕΝΤΡΙΚΗ ΒΑΣ. ΤΕΡΜΑΤΩΝ ΤΟΥ ΚΛΑΔΟΥ ΤΗΣ Α. ΚΗΦΙΣΟΥ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΔΑΜΙΑ
Wed Feb 01 02:00:00 EET 2023	MS872	Α. ΚΗΦΙΣΟΥ	206	37,85	ΡΑΜΠΑ ΕΙΣΟΔΟΥ ΠΡΟΣ ΔΥΡΡΑΧΙΟΥ / ΚΟΝΣΤΑΝΤΙΝΟΥΠΟΛΕΩΣ ΤΟΥ ΚΛΑΔΟΥ ΤΗΣ Α. ΚΗΦΙΣΟΥ ΜΕ ΚΑΤΕΥΘΥΝΣΗ
Wed Feb 01 02:00:00 EET 2023	MS866	Α. ΚΗΦΙΣΟΥ	575	51,625	ΡΑΜΠΑ ΕΙΣΟΔΟΥ ΠΡΟΣ Π. ΡΑΛΛΗ ΤΟΥ ΚΛΑΔΟΥ ΤΗΣ Α. ΚΗΦΙΣΟΥ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΠΕΙΡΑΙΑ (ΝΑ)
Wed Feb 01 02:00:00 EET 2023	MS843	Α. ΚΗΦΙΣΟΥ	66	21,5	ΡΑΜΠΑ ΕΙΣΟΔΟΥ ΠΡΟΣ Π. ΡΑΛΛΗ ΤΟΥ ΚΛΑΔΟΥ ΤΗΣ Α. ΚΗΦΙΣΟΥ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΔΑΜΙΑ 25 Μ. ΠΡΙΝ ΤΗΝ Π.
Wed Feb 01 02:00:00 EET 2023	MS807	Π. ΡΑΛΛΗ	381	18,55	ΚΥΡΙΟΣ ΔΡΟΜΟΣ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΔΑΜΙΑ 25 Μ. ΠΡΙΝ ΑΠΟ ΤΗ Α. ΚΗΦΙΣΟΥ (ΜΟΝΟ ΓΙΑ Φ.2)
Wed Feb 01 02:00:00 EET 2023	MS868	Α. ΒΟΥΛΙΑΜΕΝΗΣ	468	78,02831579	ΚΥΡΙΟΣ ΔΡΟΜΟΣ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΚΕΝΤΡΙΟ, 110 Μ. ΠΡΙΝ ΑΠΟ ΤΗΝ ΠΡΩΤΗ ΚΥΚΛΟΣ ΠΕΤΡΟΥ
Wed Feb 01 02:00:00 EET 2023	MS690	ΚΥΚΛΟΣ	30	65	ΚΥΡΙΑ ΟΔΟΣ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΠΡΟΣ Α. ΑΘΗΝΩΝ (Β). 80Μ ΠΡΙΝ ΤΗ Α. ΑΘΗΝΩΝ.
Wed Feb 01 02:00:00 EET 2023	MS646	ΑΘΗΝΩΝ	715	59,75	ΚΥΡΙΟΣ ΔΡΟΜΟΣ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΔΑΜΙΑ 150 Μ. ΠΡΙΝ ΑΠΟ ΤΗ ΦΑΘΕΡΙΟΥ
Wed Feb 01 02:00:00 EET 2023	MS644	ΑΘΗΝΩΝ	994	71,875	ΚΥΡΙΟΣ ΔΡΟΜΟΣ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΔΑΜΙΑ 150 Μ. ΠΡΙΝ ΑΠΟ ΤΗΝ ΕΥΡΥΤΑΝΙΑΣ
Wed Feb 01 02:00:00 EET 2023	MS634	ΑΘΗΝΩΝ	666	55,525	ΚΥΡΙΟΣ ΔΡΟΜΟΣ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΚΕΝΤΡΙΟ (ΑΔΙΑΜΕΣΟ) 150 Μ. ΠΡΙΝ ΑΠΟ ΤΗΝ ΣΥΝΟΡΗ ΓΙΑΤΣΗ
Wed Feb 01 02:00:00 EET 2023	MS494	ΔΥΡΡΑΧΙΟΥ	183	26,525	ΚΥΡΙΟΣ ΔΡΟΜΟΣ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΠΕΡΙΣΤΕΡ, 40 Μ. ΠΡΙΝ ΤΗ Α. ΚΗΦΙΣΟΥ
Wed Feb 01 02:00:00 EET 2023	MS362	ΑΜΑΛΙΑΣ	0	0	ΚΥΡΙΟΣ ΔΡΟΜΟΣ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΚΕΝΤΡΙΟ 100 Μ. ΠΡΙΝ ΑΠΟ ΤΗ ΒΑΣ. ΣΟΒΙΑΣ
Wed Feb 01 02:00:00 EET 2023	MS296	Α. ΚΗΦΙΣΟΥ	91	40	ΡΑΜΠΑ ΕΙΣΟΔΟΥ ΑΠΟ ΤΗ ΣΥΝΟΡΗ ΑΘΗΝΩΝ ΕΚΤΟΣ ΤΟΥ ΚΛΑΔΟΥ ΤΗΣ Α. ΚΗΦΙΣΟΥ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΚΗΦΙΔΑ
Wed Feb 01 02:00:00 EET 2023	MS287	ΚΑΠΟΔΙΣΤΡΙΟΥ	345	68,05	ΚΥΡΙΟΣ ΔΡΟΜΟΣ 150 Μ. ΠΡΙΝ ΑΠΟ ΤΗ Α. ΚΗΦΙΣΟΥ
Wed Feb 01 02:00:00 EET 2023	MS151	ΠΟΛΕΥΣΙΩΝΟΣ	92	72,3	ΚΑΤΟ ΑΠΟ ΤΗΝ ΚΗΦΙΣΟΥ, ΡΑΜΠΑ ΕΙΣΟΔΟΥ ΤΗΣ Α. ΠΟΛΕΥΣΙΩΝΟΣ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΠΡΟΣ ΤΟ ΓΗΤ. ΚΑΡΑΒΙ
Wed Feb 01 02:00:00 EET 2023	MS129	Α. ΚΗΦΙΣΟΥ	560	94,425	ΡΑΜΠΑ ΕΙΣΟΔΟΥ ΑΠΟ Α. ΠΟΛΕΥΣΙΩΝΟΣ ΑΠΟ ΓΙΑΡΑΔΑ ΣΤΗ Α. ΚΗΦΙΣΟΥ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΔΑΜΙΑ
Wed Feb 01 02:00:00 EET 2023	MS119	Α. ΚΗΦΙΣΟΥ	199	79,975	ΡΑΜΠΑ ΕΙΣΟΔΟΥ ΑΠΟ ΠΕΙΡΑΙΑ ΣΤΟΝ ΚΛΑΔΟ ΤΗΣ Α. ΚΗΦΙΣΟΥ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΔΑΜΙΑ
Wed Feb 01 02:00:00 EET 2023	MS113	Α. ΚΗΦΙΣΟΥ	81	42,6	ΡΑΜΠΑ ΕΙΣΟΔΟΥ ΑΠΟ ΑΓ. ΑΝΝΗΣ ΣΤΗ Α. ΚΗΦΙΣΟΥ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΠΕΙΡΑΙΑ
Wed Feb 01 02:00:00 EET 2023	MS111	Α. ΚΗΦΙΣΟΥ	157	56,6	ΡΑΜΠΑ ΕΙΣΟΔΟΥ ΠΡΟΣ ΑΓ. ΑΝΝΗΣ ΤΟΥ ΚΛΑΔΟΥ ΤΗΣ Α. ΚΗΦΙΣΟΥ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΔΑΜΙΑ
Wed Feb 01 02:00:00 EET 2023	MS886	ΛΕΝΟΡΜΑΝ	244	26,825	ΚΥΡΙΟΣ ΔΡΟΜΟΣ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΠΡΟΣ ΠΕΡΙΣΤΕΡ, 40 ΜΕΤΡΑ ΠΡΙΝ ΤΗ ΣΥΝΟΡΗ ΤΗΣ ΜΕ ΤΟΝ ΚΗΦΙΣΟ (Δ)
Wed Feb 01 02:00:00 EET 2023	MS864	ΙΕΡΑ ΟΔΟΣ	253	36,9	ΚΥΡΙΟΣ ΔΡΟΜΟΣ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΔΑΜΙΑ 40 Μ. ΠΡΙΝ ΑΠΟ ΤΗ Α. ΚΗΦΙΣΟΥ (ΜΟΝΟ ΓΙΑ Φ.2)
Wed Feb 01 02:00:00 EET 2023	MS860	ΠΑΡΑΛΟΡΜΟΣ Α. ΚΗΦΙΣΟΥ	47	56,3	ΕΙΣΟΔΟΣ ΑΠΟ ΠΑΡΑΛΟΡΜΟ ΤΗΣ Α. ΚΗΦΙΣΟΥ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΠΕΙΡΑΙΑ ΠΡΟΣ ΤΗΝ ΙΕΡΑ ΟΔΟ, 150 Μ. ΠΡΙΝ
Wed Feb 01 02:00:00 EET 2023	MS876	ΕΘΝ. ΜΑΚΑΡΙΟΥ	455	42,825	ΚΥΡΙΟΣ ΔΡΟΜΟΣ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΠΡΟΣ ΑΘΗΝΑ, 180 ΜΕΤΡΑ ΠΡΙΝ ΤΗ ΣΥΝΟΡΗ ΤΗΣ ΜΕ ΤΟΝ ΚΗΦΙΣΟ
Wed Feb 01 02:00:00 EET 2023	MS875	ΑΘΗΝΩΝ (ΠΑΡΑΛΟΡΜΟΣ)	213	57	ΡΑΜΠΑ ΕΙΣΟΔΟΥ ΑΠΟ Α. ΚΗΦΙΣΟΥ ΣΤΟΝ ΚΛΑΔΟ ΤΗΣ Α. ΑΘΗΝΩΝ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΚΕΝΤΡΙΟ ΠΡΙΝ ΑΠΟ ΤΗ
Wed Feb 01 02:00:00 EET 2023	MS873	Α. ΚΗΦΙΣΟΥ	459	38,6	ΡΑΜΠΑ ΕΙΣΟΔΟΥ ΠΡΟΣ ΚΟΝΣΤΑΝΤΙΝΟΥΠΟΛΕΩΣ / ΔΥΡΡΑΧΙΟΥ ΤΟΥ ΚΛΑΔΟΥ ΤΗΣ Α. ΚΗΦΙΣΟΥ ΜΕ ΚΑΤΕΥΘΥΝΣΗ

Σχήμα C.18: Αποτέλεσμα λήψης αρχείου Excel

Όσο αφορά το XML (Σχήμα C.19) και Json (Σχήμα C.20) αρχείο η μορφοποίηση αποτελείται κυρίως από την στοίχιση του περιεχομένου, παρακάτω μπορούμε να μελετήσουμε την μορφή

που έχουν τα αρχεία αυτά.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<TrafficMonitoring>
  <Measurement timestamp="Wed Feb 01 02:00:00 EET 2023">
    <sensor>MS05231/sensor</sensor>
    <averageSpeed>26.3</averageSpeed>
    <countedCars>108</countedCars>
    <roadName>Α. ΚΗΦΙΣΟΥ</roadName>
    <roadInfo>ΡΑΜΠΑ ΕΙΣΟΔΟΥ ΠΡΟΣ ΕΕΡΑ ΟΔΟ 122 ΚΛΑΔΟΥ ΤΗΣ Α. ΚΗΦΙΣΟΥ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΑΦΑΝΗ, 40 Μ. ΠΡΙΝ ΑΠΟ ΤΗΝ ΕΕΡΑ ΟΔΟ (ΚΗΦΙΣΟΣ ΓΙΑ Ο/ΣΙ)/roadInfo</roadInfo>
  </Measurement>
  <Measurement timestamp="Wed Feb 01 02:00:00 EET 2023">
    <sensor>MS0781/sensor</sensor>
    <averageSpeed>39.425</averageSpeed>
    <countedCars>68</countedCars>
    <roadName>Α. ΚΗΦΙΣΟΥ</roadName>
    <roadInfo>ΡΑΜΠΑ ΕΙΣΟΔΟΥ ΠΡΟΣ ΛΕΝΟΡΜΑΝ / ΒΑΣ. ΓΕΩΡΓΙΟΥ 122 ΚΛΑΔΟΥ ΤΗΣ Α. ΚΗΦΙΣΟΥ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΑΦΑΝΗ 180 Μ. ΠΡΙΝ ΑΠΟ ΤΗ ΛΕΝΟΡΜΑΝ</roadInfo>
  </Measurement>
  <Measurement timestamp="Wed Feb 01 02:00:00 EET 2023">
    <sensor>MS0722/sensor</sensor>
    <averageSpeed>37.59</averageSpeed>
    <countedCars>206</countedCars>
    <roadName>Α. ΚΗΦΙΣΟΥ</roadName>
    <roadInfo>ΡΑΜΠΑ ΕΙΣΟΔΟΥ ΠΡΟΣ ΔΥΡΡΑΧΙΟΥ / ΚΩΝΣΤΑΝΤΙΝΟΥΔΑΚΕ 122 ΚΛΑΔΟΥ ΤΗΣ Α. ΚΗΦΙΣΟΥ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΑΦΑΝΗ 185 Μ. ΠΡΙΝ ΑΠΟ ΤΗ ΔΥΡΡΑΧΙΟΥ</roadInfo>
  </Measurement>
  <Measurement timestamp="Wed Feb 01 02:00:00 EET 2023">
    <sensor>MS055</sensor>
    <averageSpeed>33.252</averageSpeed>
    <countedCars>578</countedCars>
    <roadName>Α. ΚΗΦΙΣΟΥ</roadName>
    <roadInfo>ΡΑΜΠΑ ΕΙΣΟΔΟΥ ΠΡΟΣ Π. ΠΑΛΗ 122 ΚΛΑΔΟΥ ΤΗΣ Α. ΚΗΦΙΣΟΥ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΠΕΙΡΑΙΑ (ΝΔ)/roadInfo</roadInfo>
  </Measurement>
</TrafficMonitoring>
```

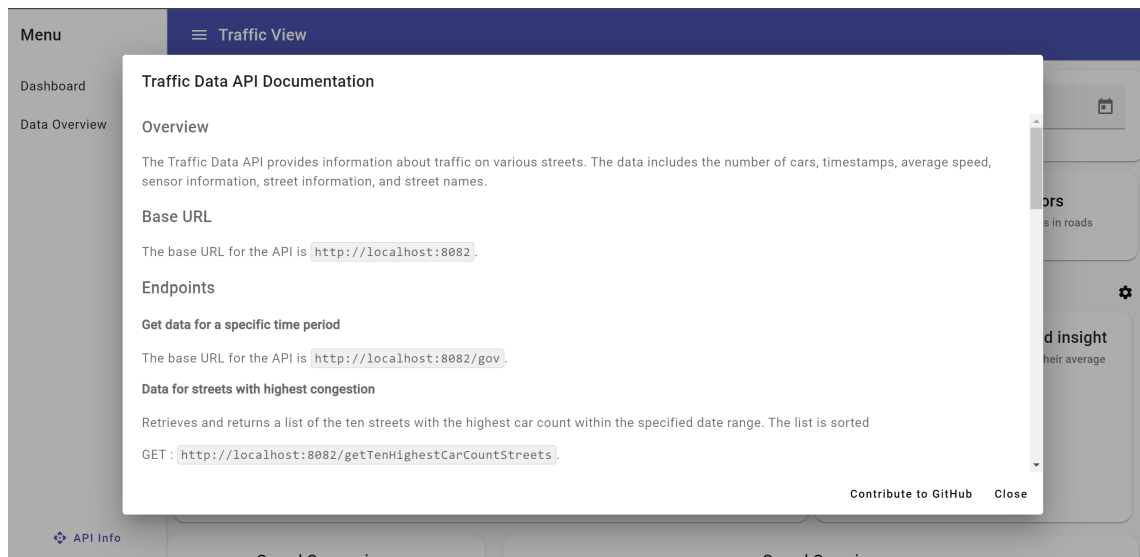
```
{
  "road_name": "ΕΘΝ. ΜΑΚΑΡΙΟΥ",
  "countedCars": 138,
  "averageSpeed": 21,
  "sensor": "MS088",
  "roadInfo": "ΤΟΥΡΙΣΤΙΚΟ ΔΡΟΜΟ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΠΡΟΣ ΑΦΑΝΗ, 25 ΜΕΤΡΑ ΠΡΙΝ ΤΗΝ ΣΥΝΘΥΣΗ ΤΗΣ ΜΕ ΤΟΝ ΚΗΦΙΣΟ (ΚΗΦΙΣΟΣ ΓΙΑ Ο/ΣΙ)",
  "timestamp": "Wed Feb 01 02:00:00 EET 2023"
},
{
  "road_name": "Α. ΚΗΦΙΣΟΥ",
  "countedCars": 108,
  "averageSpeed": 26.3,
  "sensor": "MS0523",
  "roadInfo": "ΡΑΜΠΑ ΕΙΣΟΔΟΥ ΠΡΟΣ ΕΕΡΑ ΟΔΟ ΤΟΥ ΚΛΑΔΟΥ ΤΗΣ Α. ΚΗΦΙΣΟΥ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΑΦΑΝΗ 40 Μ. ΠΡΙΝ ΑΠΟ ΤΗΝ ΕΕΡΑ ΟΔΟ (ΚΗΦΙΣΟΣ ΓΙΑ Ο/ΣΙ)",
  "timestamp": "Wed Feb 01 02:00:00 EET 2023"
},
{
  "road_name": "Α. ΚΗΦΙΣΟΥ",
  "countedCars": 68,
  "averageSpeed": 39.425,
  "sensor": "MS0781",
  "roadInfo": "ΡΑΜΠΑ ΕΙΣΟΔΟΥ ΠΡΟΣ ΛΕΝΟΡΜΑΝ / ΒΑΣ. ΓΕΩΡΓΙΟΥ ΤΟΥ ΚΛΑΔΟΥ ΤΗΣ Α. ΚΗΦΙΣΟΥ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΑΦΑΝΗ 180 Μ. ΠΡΙΝ ΑΠΟ ΤΗ ΛΕΝΟΡΜΑΝ",
  "timestamp": "Wed Feb 01 02:00:00 EET 2023"
},
{
  "road_name": "Α. ΚΗΦΙΣΟΥ",
  "countedCars": 206,
  "averageSpeed": 37.59,
  "sensor": "MS0722",
  "roadInfo": "ΡΑΜΠΑ ΕΙΣΟΔΟΥ ΠΡΟΣ ΔΥΡΡΑΧΙΟΥ / ΚΩΝΣΤΑΝΤΙΝΟΥΔΑΚΕ ΤΟΥ ΚΛΑΔΟΥ ΤΗΣ Α. ΚΗΦΙΣΟΥ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΑΦΑΝΗ 185 Μ. ΠΡΙΝ ΑΠΟ ΤΗ ΔΥΡΡΑΧΙΟΥ",
  "timestamp": "Wed Feb 01 02:00:00 EET 2023"
},
{
  "road_name": "Α. ΚΗΦΙΣΟΥ",
  "countedCars": 578,
  "averageSpeed": 33.252,
  "sensor": "MS055",
  "roadInfo": "ΡΑΜΠΑ ΕΙΣΟΔΟΥ ΠΡΟΣ ΔΥΡΡΑΧΙΟΥ / ΚΩΝΣΤΑΝΤΙΝΟΥΔΑΚΕ ΤΟΥ ΚΛΑΔΟΥ ΤΗΣ Α. ΚΗΦΙΣΟΥ ΜΕ ΚΑΤΕΥΘΥΝΣΗ ΑΦΑΝΗ 185 Μ. ΠΡΙΝ ΑΠΟ ΤΗ ΔΥΡΡΑΧΙΟΥ",
  "timestamp": "Wed Feb 01 02:00:00 EET 2023"
}
```

Σχήμα C.19: Αποτέλεσμα αρχείου XML

Σχήμα C.20: Αποτέλεσμα αρχείου JSON

C.1.3 API Info

Καταλήγοντας στην παρουσίαση της εφαρμογής, είναι σημαντικό να αναφερθεί μια πρόσθετη λειτουργία που προσφέρει. Ο χρήστης, μέσω του μενού, έχει τη δυνατότητα να επιλέξει το κουμπί "API Info" και να προβάλει λεπτομερώς την τεκμηρίωση των Rest API που υπάρχουν στο backend της εφαρμογής, με εξηγήσεις και παραδείγματα. Επιπλέον, παρέχεται η επιλογή ανακατεύθυνσης στο κεντρικό αποθετήριο της εφαρμογής στο GitHub (URL: <https://github.com/p18vogd/StreetTrafficMonitoringApplication>) (Σχήμα C.21), όπου όλος ο κώδικας της εφαρμογής είναι διαθέσιμος, μαζί με λεπτομερείς οδηγίες για την εγκατάσταση της εφαρμογής και τις απαιτήσεις της. Αυτή η λειτουργία απευθύνεται κυρίως σε προγραμματιστές και δημιουργούς λογισμικού που σκοπεύουν να χρησιμοποιήσουν και να ενσωματώσουν ένα τμήμα της εφαρμογής στη δική τους εφαρμογή, δημιουργώντας έτσι μια intergradable εφαρμογή. Αυτή η δυνατότητα μπορεί να πραγματοποιηθεί μέσω της χρήσης του Docker, όπου μπορούμε να χρησιμοποιήσουμε την εικόνα για τη δημιουργία άλλων εφαρμογών.



Σχήμα C.21: Παρουσίαση επεξήγησης API endpoints μέσω της εφαρμογής

Κεφάλαιο D

Υλοποίηση Εφαρμογής

Η ανάπτυξη λογισμικού είναι μια σύνθεση πολλών στοιχείων που αλληλοεπιδρούν μεταξύ τους με αρμονία και αποτελεσματικότητα. Σε αυτό το κεφάλαιο, θα εξερευνήσουμε την υλοποίηση της εφαρμογής καθώς θα μελετήσουμε το πώς χρησιμοποιήθηκαν οι τεχνολογίες Spring Boot, Angular και Docker. Θα εξετάσουμε το πώς μπορούμε να αξιοποιήσουμε την ευελιξία του Spring Boot για την ανάπτυξη ενός αποτελεσματικού και ευέλικτου backend, τη δυνατότητα της Angular να δημιουργήσουμε ένα δυναμικό και χρήσιμο frontend, καθώς και τη δύναμη του Docker για τη δημιουργία και τη διαχείριση εφαρμογών σε περιβάλλοντα εκτέλεσης που βασίζονται σε containers. Αυτή η συνδυαστική προσέγγιση επιτρέπει την ανάπτυξη μιας πλήρως λειτουργικής και ευέλικτης εφαρμογής που μπορεί να ανταποκριθεί στις σύγχρονες απαιτήσεις.

D.1 Backend

Ο πυρήνας της εφαρμογής, γνωστός και ως backend αποτελεί τον λειτουργικό άξονα πίσω από τη λειτουργία και τη δυναμική της, διαχειρίζεται την επικοινωνία με τα εξωτερικά API για την λήψη δεδομένων, την διαχείριση και επεξεργασία τους καθώς και την αποτελεσματική διαχείριση των αιτημάτων από το frontend. Πιο αναλυτικά στην ενότητα αυτή θα μελετήσουμε τις πέντε βασικές πτυχές του backend. Οι πτυχές αυτές αποτελούνται από το Spring Boot που απλοποιεί την ανάπτυξη εφαρμογών, την αρχιτεκτονική backend που καθορίζει τη βάση του συστήματος, τη διαχείριση εξαρτήσεων για τη διασφάλιση της συνέπειας και της ευστάθειας, τη συλλογή δεδομένων που αποτελεί τον πυρήνα των πληροφοριακών συστη-

μάτων, και τον σχεδιασμό API για την αποτελεσματική επικοινωνία μεταξύ των συστατικών του συστήματος. συστήματος.

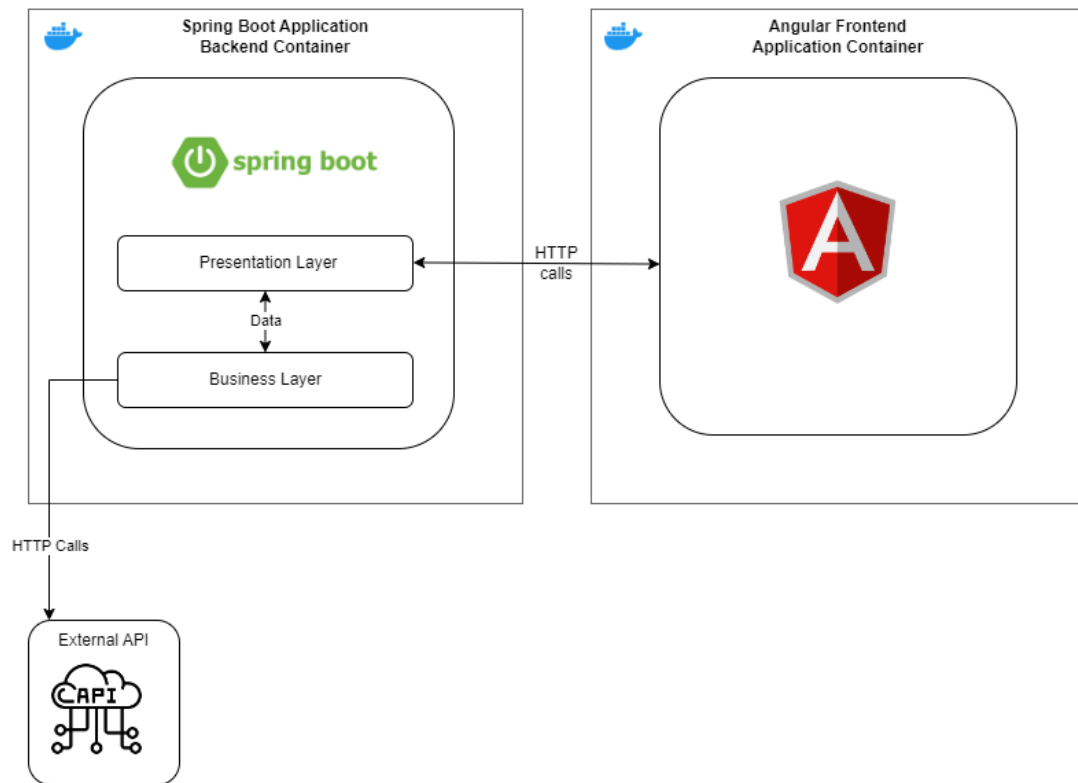
D.1.1 Spring Boot

Στο επόμενο υποκεφάλαιο, θα εξετάσουμε το backend της εφαρμογής, την αρχιτεκτονική του και τον τρόπο λειτουργίας του. Το backend είναι υλοποιημένο σε Spring Boot, ένα ισχυρό πλαίσιο για τη δημιουργία εφαρμογών Java. Ως μια αντικειμενοστραφής γλώσσα προγραμματισμού η Java είναι ευρέως διαδεδομένη και χρησιμοποιείται σε πολλές εφαρμογές, από επιχειρηματικά συστήματα μέχρι κινητές εφαρμογές. Παράλληλα σε συνδυασμό με το Spring Boot επιτρέπει την εύελικτη και ασφαλή διαχείριση των δεδομένων, την εύκολη επέκταση και την ομαλή ενσωμάτωση με άλλα συστήματα και υπηρεσίες.

D.1.1.1 Αρχιτεκτονική Backend

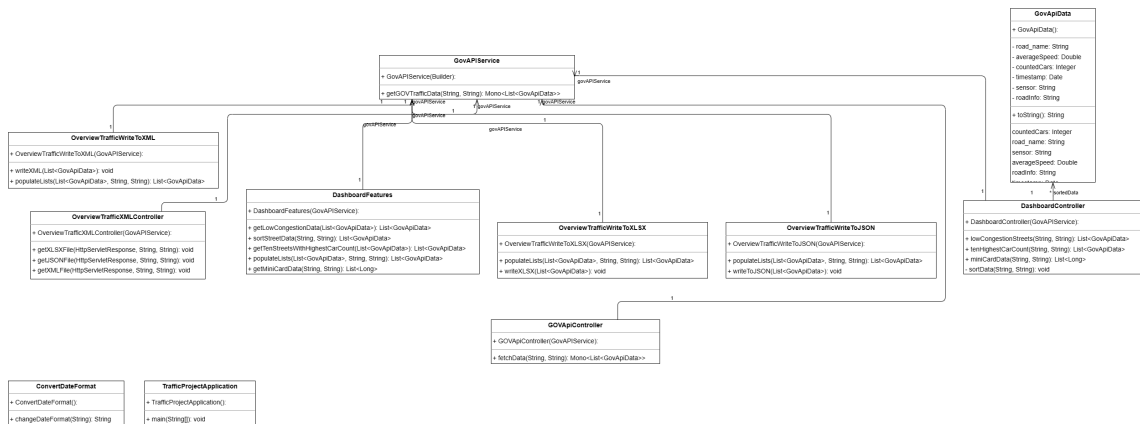
Η εφαρμογή που αναπτύχθηκε αποτελεί ένα πολυεπίπεδο σύστημα, το οποίο ενσωματώνει ένα backend σε Spring Boot και ένα frontend σε Angular. Κάθε εφαρμογή λειτουργεί ανεξάρτητα μέσα σε δικά της containers. Εξετάζοντας πιο βαθιά την αρχιτεκτονική του backend, διαπιστώνουμε ότι αποτελείται από δύο επίπεδα το Business Layer και το Presentation Layer. Το Business Layer είναι υπεύθυνο για την επεξεργασία και λήψη των δεδομένων που λαμβάνονται από το εξωτερικό API του gov. Εδώ γίνεται η λογική επεξεργασία των δεδομένων, όπως η ταξινόμηση, το φιλτράρισμα, οι μετατροπές και άλλες επεξεργασίες που είναι απαραίτητες για την εφαρμογή. Το Presentation Layer είναι υπεύθυνο για την επικοινωνία με το frontend. Οι controllers λαμβάνουν τα επεξεργασμένα δεδομένα από το Business Layer και τα στέλνουν στο frontend μέσω του REST API. Σκοπός των επιπέδων αυτών είναι η καθαρή διάκριση μεταξύ της λογικής επεξεργασίας και της παρουσίασης των δεδομένων.

Η Angular εφαρμογή τρέχει σε ένα ξεχωριστό container. Αυτό το στοιχείο της εφαρμογής είναι αρμόδιο για την οπτικοποίηση των δεδομένων στον χρήστη. Πιο συγκεκριμένα, πρόκειται για μια ανεξάρτητη εφαρμογή που καταναλώνει τα REST API του backend για να αποκτήσει πρόσβαση στα δεδομένα και στη συνέχεια, τα παρουσιάζει στον χρήστη με έναν ευανάγνωστο τρόπο μέσα από μια γραφική διεπαφή UI.



Σχήμα D.22: Αρχιτεκτονική Backend εφαρμογής

Στο διάγραμμα που ακολουθεί, παρουσιάζεται ο κώδικας του Backend που θα αναλυθεί στην συνέχεια του κεφαλαίου. Μέσα από αυτό το διάγραμμα μπορούμε να δούμε αναλυτικά τις μεθόδους που περιλαμβάνει η κάθε κλάση καθώς και τον τρόπο που επικοινωνούν ανά τα διαφορετικά επίπεδα της εφαρμογής. Παρατηρούμε ότι σε κάθε επίπεδο της εφαρμογής που έχουμε περιγράψει, η κλάση `GonAPIService` διαδραματίζει κεντρικό ρόλο. Αυτή η κλάση είναι υπεύθυνη για την παροχή των δεδομένων που απαιτούνται για την ομαλή λειτουργία της εφαρμογής.



Σχήμα D.23: Αναπαράσταση του κώδικα Backend σε διάγραμμα

D.1.1.2 Dependency Management

Παρακάτω θα μελετήσουμε το τρόπο με τον οποίο διαχειρίστηκαν οι εξαρτήσεις της εφαρμογής. Η διαχείριση έγινε μέσω του Maven το οποίο λειτουργεί ως εργαλείο ρύθμισης και περιέχει πληροφορίες σχετικά με την εφαρμογή, τις εξαρτήσεις που χρησιμοποιήθηκαν και τις ρυθμίσεις για το build της εφαρμογής μας. Με άλλα λόγια θα περιγράψουμε τον τρόπο με τον οποίο χειρίστηκαν όλα τα εξωτερικά πακέτα και βιβλιοθήκες που ήταν απαραίτητες για την ανάπτυξη της εφαρμογής.

Στο κεντρικό pom.xml της εφαρμογής μας πάνω περιγράφονται οι βασικές πληροφορίες του project. Αυτές περιλαμβάνουν το Parent Project το οποίο είναι το spring-boot-starter-parent συνοδευόμενο με την έκδοση του Spring Boot που είναι 3.1.5 και κάποιες πληροφορίες για την έκδοση της εφαρμογής μας που είναι η 0.0.1-SNAPSHOT. Το packaging μας δηλώνει τον τύπο του project. Πιο αναλυτικά στο κεντρικό pom.xml που αναλύουμε, το packaging έχει πάρει την τιμή pom. Αυτό σημαίνει κατά το build δεν θα παραχθεί κάποιο εκτελέσιμο αρχείο, αλλά μπορεί να περιέχει πληροφορίες που χρησιμοποιούνται από άλλα υποέργα για τη διαχείριση των εξαρτήσεών τους. Η προσέγγιση του να έχουμε πολλά pom αρχεία είναι χρήσιμη όταν το έργο είναι πολυ-επίπεδο με πολλά άλλα υποέργα. Το κεντρικό αυτό POM θα χρησιμοποιηθεί για τη διαχείριση των εξαρτήσεων, των ρυθμίσεων του build και άλλων κοινών ρυθμίσεων σε ολόκληρο το έργο.

Listing D.1: Περιγραφή του pom

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://↵
  www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://↵
  maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>3.1.5</version>
9     <relativePath/> <!-- lookup parent from repository -->
10  </parent>
11  <groupId>com.example</groupId>
12  <artifactId>TrafficProject</artifactId>
13  <version>0.0.1-SNAPSHOT</version>
14  <packaging>pom</packaging>
15  <name>TrafficProject</name>
16  <description>Spring Boot App for street traffic monitoring </↵
    description>
17 </project>

```

Μέσα στα properties καθορίζεται και η έκδοση της Java που χρησιμοποιήθηκε. Το κομμάτι του module περιλαμβάνει όλα τα υποέργα που περιέχει το project. Στην εφαρμογή έχουμε ένα υποέργο με το όνομα backend. Το υποέργο αυτό περιλαμβάνει το δικό του pom xml καθώς και όλο τον πηγαίο κώδικα της backend εφαρμογής μας

Listing D.2: Περιγραφή των sub-modules και των properties

```

1   <modules>
2     <module>backend</module>
3   </modules>
4   <properties>
5     <java.version>17</java.version>
6   </properties>

```

Παρακάτω περιγράφονται όλες οι εξωτερικές βιβλιοθήκες και πακέτα που χρησιμοποιήθηκαν. Για να γίνει χρήση αυτών προσθέτουμε το παρακάτω τμήμα στο κεντρικό μας pom και το

maven θα αναλάβει να τις κατεβάσει από το κεντρικό αποθετήριο του. Πιο αναλυτικά τα dependencies που χρησιμοποιήθηκαν είναι τα εξής:

- **spring-boot-starter-web:** Αυτή η εξάρτηση παρέχει όλα τα απαραίτητα για τη δημιουργία μιας web εφαρμογής, συμπεριλαμβανομένων των RESTful, JSON και των Tomcat ως ενσωματωμένος server.
- **spring-boot-docker-compose:** Αυτή η εξάρτηση επιτρέπει την ενσωμάτωση της εφαρμογής με το Docker Compose, το οποίο είναι ένα εργαλείο για τον ορισμό και τη διαχείριση multi-container Docker εφαρμογών.
- **spring-boot-starter-webflux:** Αυτή η εξάρτηση επιτρέπει την δημιουργία reactive εφαρμογών, οι οποίες είναι ικανές να χειριστούν μεγάλο αριθμό αιτήσεων ασύγχρονα.
- **reactor-spring:** Αυτή η εξάρτηση παρέχει την υποστήριξη για την ανάπτυξη reactive εφαρμογών με τη χρήση του Spring Framework.
- **json:** Αυτή η εξάρτηση επιτρέπει την επεξεργασία JSON δεδομένων στην εφαρμογή σας.
- **poi-ooxml:** Αυτή η εξάρτηση παρέχει τη δυνατότητα για την διαχείριση Excel αρχείων χρησιμοποιώντας την Java.

Listing D.3: *Dependencies εφαρμογής*

```

1 <dependencies>
2     <dependency>
3         <groupId>org.springframework.boot</groupId>
4         <artifactId>spring-boot-starter-web</artifactId>
5     </dependency>
6     <dependency>
7         <groupId>org.springframework.boot</groupId>
8         <artifactId>spring-boot-docker-compose</artifactId>
9         <scope>runtime</scope>
10        <optional>true</optional>
11    </dependency>
12    <dependency>
13        <groupId>org.springframework.boot</groupId>
14        <artifactId>spring-boot-starter-webflux</artifactId>
15    </dependency>
16    <dependency>
17        <groupId>org.projectreactor</groupId>
18        <artifactId>reactor-spring</artifactId>
19        <version>1.0.1.RELEASE</version>
20    </dependency>
21    <dependency>
22        <groupId>org.json</groupId>
23        <artifactId>json</artifactId>
24        <version>20230618</version>
25    </dependency>
26    <dependency>
27        <groupId>org.apache.poi</groupId>
28        <artifactId>poi-ooxml</artifactId>
29        <version>4.1.2</version>
30    </dependency>
31    <dependency>
32        <groupId>org.springframework.boot</groupId>
33        <artifactId>spring-boot-starter-test</artifactId>
34        <scope>test</scope>
35    </dependency>
36 </dependencies>

```

Αυτό το τμήμα του POM XML αρχείου περιγράφει τη διαδικασία build της εφαρμογής. Χρησιμοποιεί το spring-boot-maven-plugin, το οποίο είναι ένα plugin του Maven που παρέχει από το Spring Boot. Αυτό το plugin μας παρέχει την δυνατότητα να δημιουργούμε εκτελέσιμα JARs και Docker images. Στην ενότητα configuration, ορίζετε το paketobuildpacks/builder-jammy-base:latest ως builder για τη δημιουργία της Docker image. Τα Paketo Buildpacks είναι μια συλλογή από Cloud Native Buildpacks που επιτρέπουν τη δημιουργία εικόνων που μπορούν να τρέξουν σε οποιαδήποτε cloud platform.

Listing D.4: Διαμόρφωση του τρόπου χησιμάτος της εφαρμογής

```

1    <build>
2        <plugins>
3            <plugin>
4                <groupId>org.springframework.boot</groupId>
5                <artifactId>spring-boot-maven-plugin</artifactId>
6                <configuration>
7                    <image>
8                        <builder>paketobuildpacks/builder-jammy-↵
                            base:latest</builder>
9                    </image>
10                </configuration>
11            </plugin>
12        </plugins>
13    </build>

```

D.1.1.3 Συλλογή δεδομένων

Τα δεδομένα που ενσωματώνονται στην εφαρμογή προέρχονται από το API της Κυβέρνησης (GOV), το οποίο μας παρέχει λεπτομερείς πληροφορίες σχετικά με τους δρόμους της Αθήνας. Αρχή των δεδομένων καθορίζεται ο Νοέμβριος του 2020 και έκτοτε τα δεδομένα που παρέχει ανανεώνονται με συχνό ρυθμό και φτάνουν μέχρι και τέλη Ιανουαρίου του 2024. Ως τρόπο απόκτησης των δεδομένων αυτών θα χρειαστεί να καθορίσουμε μια χρονική περίοδο που την δίνουμε με μορφή παραμέτρων και ένα API token το οποίο μας αποστέλλετε αφού έχουμε ολοκληρώσει την εγγραφή στο data.gov.gr. Ως απάντηση σε κάθε GET κλήση στο API λαμβάνουμε ένα JSON που περιλαμβάνει την εξής πληροφορία:

- **Ακριβής χρονική στιγμή καταγραφής:** Η συγκεκριμένη στιγμή που τα δε-

δομένα καταγράφηκαν.

- **Όνομα δρόμου:** Το όνομα του δρόμου που εξετάζεται.
- **Πλήθος οχημάτων:** Ο αριθμός των οχημάτων που καταγράφηκαν στον συγκεκριμένο δρόμο κατά τη δεδομένη χρονική στιγμή.
- **Μέση ταχύτητα οχημάτων:** Η μέση ταχύτητα των καταγεγραμμένων οχημάτων.
- **Όνομα αισθητήρα:** Το όνομα του αισθητήρα που πραγματοποίησε την καταγραφή.

Listing D.5: *Backend Web Client για την συλλογή δεδομένων από API*

```

1
2 @Service
3 public class GovAPIService {
4
5     @Value("${gov.api.key}")
6     private String apiKey;
7
8     private final WebClient webClient;
9
10    public GovAPIService(WebClient.Builder webClientBuilder ){
11        this.webClient = webClientBuilder.baseUrl("https://data.gov.↵
12        gr").build();
13    }
14    public Mono<List<GovApiData>> getGOVTrafficData(String startDate,↵
15        String endDate){
16        return this.webClient.get()
17            .uri(uriBuilder -> uriBuilder
18                .path("/api/v1/query/road_traffic_attica")
19                .queryParams("date_from",startDate)
20                .queryParams("date_to",endDate)
21                .build())
22            .header("Authorization",apiKey)
23            .retrieve()
24            .bodyToFlux(GovApiData.class)
25            .collectList();
26    }
27 }
```

Μετά την παροχή των απαραίτητων παραμέτρων στο API, πραγματοποιούμε μια GET κλήση και λαμβάνουμε μια απάντηση σε μορφή JSON. Κατά την ολοκλήρωση της κλήσης, εκτελούμε επιπρόσθετες διαδικασίες για τον χειρισμό των ληφθέντων δεδομένων. Συγκεκριμένα, αντιστοιχίζουμε τα πεδία της ληφθείσας απάντησης JSON σε αντικείμενα της κλάσης GovApiData και τα αποθηκεύουμε σε μια λίστα. Αυτό το κάνουμε για να επιτύχουμε μεγαλύτερη ευελιξία στον χειρισμό των δεδομένων εντός της εφαρμογής, επιτρέποντάς μας να εξάγουμε πιο εύκολα πρόσθετες πληροφορίες από αυτές που παρέχει το API.

Listing D.6: Κλάση που περιγράφει τα δεδομένα της εφαρμογής

```
1 public class GovApiData {
2     @JsonProperty("road_name")
3     private String road_name;
4     @JsonProperty("road_info")
5     private String roadInfo;
6     @JsonProperty("countedcars")
7     private Integer countedCars;
8     @JsonProperty("average_speed")
9     private Double averageSpeed;
10    @JsonProperty("appprocesstime")
11    private Date timestamp;
12    @JsonProperty("deviceid")
13    private String sensor;
14
15    public Date getTimestamp() {return timestamp;}
16
17    public void setTimestamp(Date timestamp) {
18        this.timestamp = timestamp;
19    }
20
21    public Double getAverageSpeed() {
22        return averageSpeed;
23    }
24
25    public void setAverageSpeed(Double averageSpeed) {
26        this.averageSpeed = averageSpeed;
27    }
28
```

```
29     public Integer getCountedCars() {
30         return countedCars;
31     }
32
33     public void setCountedCars(Integer countedCars) {
34         this.countedCars = countedCars;
35     }
36
37     public String getRoadInfo() {
38         return roadInfo;
39     }
40
41     public void setRoadInfo(String roadInfo) {
42         this.roadInfo = roadInfo;
43     }
44
45     public String getRoad_name() {
46         return road_name;
47     }
48
49     public void setRoad_name(String road_name) {
50         this.road_name = road_name;
51     }
52
53     public String getSensor() {
54         return sensor;
55     }
56     public void setSensor(String sensor) {
57         this.sensor = sensor;
58     }
59
60     @Override
61     public String toString() {
62         return "GovApiData{" +
63             "road_name='" + road_name + '\\\' +
64             ", roadInfo='" + roadInfo + '\\\' +
65             ", countedCars=" + countedCars +
66             ", averageSpeed=" + averageSpeed +
```

```

67         ", timestamp=" + timestamp +
68         ", sensor='" + sensor + '\\\' +
69         '}' ;
70     }
71 }

```

Στο σημείο αυτό να αναφέρουμε ότι ο WebClient (Listing D.5) για τις Java εφαρμογές παρέχεται από το dependency του WebFlux, το οποίο μας προσφέρει reactive εργαλεία για την ανάπτυξη Java λογισμικού. Ένα από τα μεγαλύτερα θετικά που μας προσφέρει ο συγκεκριμένος WebClient είναι η ικανότητα του να δέχεται μεγάλο όγκο δεδομένων σε σχέση με τα Rest Templates. Αυτό το επιτυγχάνει με το να δέχεται τα δεδομένα σε ροές, επιτρέποντας την επεξεργασία τους καθώς φτάνουν, χωρίς την ανάγκη αναμονής για την ολοκλήρωση της πλήρους απάντησης. Επιπλέον είναι σχεδιασμένος ώστε να λειτουργεί ασύγχρονα, επιτρέποντας στην εφαρμογή να χειρίζεται πολλαπλά αιτήματα χωρίς να αποκλείονται. Ταυτόχρονα διαχειρίζεται τους πόρους του συστήματος μεριμνώντας για την αποτελεσματική χρήση τους και την αποφυγή διαρροών μνήμης ή άλλων προβλημάτων που μπορεί να προκύψουν λόγω της χρήσης πόρων μέσα από τεχνικές non-blocking I/O.

D.1.1.4 Διαχείριση και επεξεργασία δεδομένων

Πέρα από την λήψη των δεδομένων που αναφέραμε εξίσου σημαντική είναι και η επεξεργασία τους με σκοπό να παραγάγουμε επιπλέον πληροφορία.

Στην παρακάτω μέθοδο υπολογίζουμε τις γενικές πληροφορίες που προβάλλονται από τα τέσσερα διαδοχικά πάνελ του Dashboard. Σαν παραμέτρους δεχόμαστε τις ημερομηνίες που έχει θέσει ο θέσει ο χρήστης και επιστρέφουμε μια λίστα με τις γενικές πληροφορίες τύπου Long (Listing D.7).

Ως τρόπος λειτουργίας αυτής της μεθόδου αρχικά κάνουμε κλήση τη μέθοδο getGOVTrafficData του GovApiService για να ανακτήσουμε τα δεδομένα κυκλοφορίας μεταξύ των δύο ημερομηνιών που δόθηκαν ως παράμετροι. Αφού λάβουμε τα δεδομένα σε λίστα από αντικείμενα GovApiData υλοποιούμε διάφορα Streams τα οποία θα μας προσπελάσουν την λίστα και θα επεξεργαστούμε τα δεδομένα. Το αποτέλεσμα της επεξεργασίας τους θα το αποθηκεύσουμε σε μια μεταβλητή την οποία θα προσθέσουμε και στην λίστα που θα επιστρέψουμε. Η λίστα αυτή θα περιέχει πάντα τέσσερις θέσεις όπου κάθε θέση της θα περιέχει και την τιμή που θα προβληθεί στο Dashboard στο αρχικό τμήμα με τις γενικές πληροφορίες. Πιο

αναλυτικά οι μεταβλητές αυτές θα υπολογίζονται ως εξής:

- **numberOfVehicles:** Καθώς προσπελάσουμε την λίστα με τα δεδομένα μας κρατάμε το αριθμό από τα οχήματα για τα οποία έχουμε εγγραφή. Ο αριθμός αυτός προστίθεται σε ένα συνολικό άθροισμα.
- **averageSpeedAcrossNetwork:** Καθώς προσπελάσουμε την λίστα, κρατάμε την μέση ταχύτητα που έχει καταγραφεί, την τιμή αυτή την μετατρέπουμε σε ακέραιο και υπολογίζουμε τον μέσο όρο από όλες τις τιμές.
- **numberOfRoads:** Από τα ονόματα των δρόμων κρατάμε το πλήθος των διαφορετικών τιμών που μπορεί να υπάρχουν
- **numberOfSensors:** Κρατάμε τον πλήθος των διαφορετικών αισθητήρων που υπάρχουν στις καταγραφές μας

Listing D.7: Υπολογισμός γενικών πληροφοριών κίνησης

```

1  public List<Long> getMiniCardData(String startDate, String endDate↵
    ){
2      List<GovApiData> streetData = this.govAPIService.↵
        getGOVTrafficData(startDate, endDate).block();
3      List<Long> miniCardList = new ArrayList<>();
4
5      long numberOfVehicles = streetData.stream()
6          .map(GovApiData::getCountedCars)
7          .mapToLong(Integer::intValue).sum();
8
9      NumberFormat numberFormat = NumberFormat.getNumberInstance(↵
        Locale.GERMANY);
10     String formattedNumber = numberFormat.format(numberOfVehicles↵
        );
11     miniCardList.add(numberOfVehicles);
12     System.out.println("Number Of Vehicles passed " + ↵
        formattedNumber);
13
14     long averageSpeedAcrossNetwork = (long) streetData.stream()
15         .map(GovApiData::getAverageSpeed)

```

```

16         .mapToInt(Double::intValue)
17         .average()
18         .orElse(0);
19     miniCardList.add(averageSpeedAcrossNetwork);
20     System.out.println("Average Speed " + ←
        averageSpeedAcrossNetwork);
21
22     long numberOfRoads = (long) streetData.stream()
23         .map(GovApiData::getRoad_name)
24         .distinct()
25         .count();
26     miniCardList.add(numberOfRoads);
27     System.out.println("Number of Roads " + numberOfRoads);
28
29     long numberOfSensors = streetData.stream()
30         .map(GovApiData::getSensor)
31         .distinct()
32         .count();
33     System.out.println("Number of Sensors " + numberOfSensors);
34     miniCardList.add(numberOfSensors);
35
36     return miniCardList;
37 }

```

Η μέθοδος `sortStreetData` (Listing D.8) λαμβάνει ως είσοδο δύο παραμέτρους, `startDate` και `endDate`, οι οποίες καθορίζονται από τον χρήστη μέσα από το frontend. Χρησιμοποιείται για την ταξινόμηση δεδομένων κυκλοφορίας αυτοκινήτων από τα δεδομένα που λάβαμε από το `govAPIService` (Listing D.5). Οι επιστρεφόμενες πληροφορίες συγκρίνονται με βάση τον αριθμό των καταμετρημένων αυτοκινήτων. Αρχικά, τα δεδομένα ταξινομούνται με βάση το όνομα του δρόμου, και στη συνέχεια επιλέγεται το αντικείμενο από κάθε ομάδα (δηλαδή, από κάθε δρόμο) με το μέγιστο αριθμό αυτοκινήτων. Τέλος, τα επιλεγμένα αντικείμενα ταξινομούνται με φθίνουσα σειρά βάσει του αριθμού των καταμετρημένων αυτοκινήτων και επιστρέφεται μια λίστα. Η ανάγκη για την δημιουργία μιας τέτοιας μεθόδου προκύπτει από την φύση των δεδομένων. Μέσα από μια κλήση στο API του gov μπορεί να λάβουμε αρκετές εγγραφές που σχετίζονται με έναν συγκεκριμένο δρόμο. Επόμενο λοιπόν είναι αν στις όπτικοποιήσεις τα δεδομένα που προβάλλουμε σε διαγράμματα καθορίζονται μόνο από τον αριθμό των οχη-

μάτων, να προβάλουμε μόνο δρόμους όπως η Α. Κηφισού και Α. Βουλιαγμένης που είναι από τους μεγαλύτερους δρόμους στο οδικό δίκτυο της Αθήνας. Με αυτό τον τρόπο μπορούμε να παρέχουμε περισσότερη και πιο ακριβή πληροφορία για την κατάσταση στο οδικό δίκτυο, δεδομένου ότι στην επεξεργασία των δεδομένων λαμβάνονται υπόψη επιπλέον παράμετροι.

Listing D.8: Ταξινόμηση δεδομένων κυκλοφορίας

```

1      public List<GovApiData> sortStreetData(String startDate, String ↵
        endDate){
2          List<GovApiData> streetData = this.govAPIService.↵
            getGOVTrafficData(startDate, endDate).block();
3      return streetData.stream()
4          .collect(Collectors.groupingBy(GovApiData::↵
            getRoad_name)) // Group by road name
5          .values().stream()
6          .map(roadDatas -> roadDatas.stream()
7              .max(Comparator.comparingInt(GovApiData::↵
            getCountedCars))) // Get the road data ↵
            with the max counted cars for each road
8          .filter(Optional::isPresent)
9          .map(Optional::get)
10         .sorted((rd1, rd2) -> rd2.getCountedCars() - rd1.↵
            getCountedCars())
11         .collect(Collectors.toList());
12     }

```

Από τα δεδομένα αυτά θα κρατήσουμε τις πρώτες δέκα ταξινομημένες θέσεις της λίστας (Listing D.9), οι οποίες αποτελούν και την πληροφορία που θα προβληθεί στο Dashboard που προβάλλει του δρόμους με την μεγαλύτερη κυκλοφοριακή συμφόρηση. Την λίστα αυτή την προσπελάσουμε μέσα από ένα stream και με την μέθοδο limit καθορίζουμε ότι θα κρατήσουμε τις πρώτες δέκα θέσεις της λίστας.

Listing D.9: Φιλτράρισμα δεδομένων ώστε να κρατήσουμε τις δέκα μεγαλύτερες εγγραφές

```

1      public List<GovApiData> getTenStreetsWithHighestCarCount(List<↵
        GovApiData> streetData){
2          return streetData.stream()
3              .limit(10)
4              .collect(Collectors.toList());

```

5 }

Αντίστοιχη λογική ακολουθούμε για να συλλέξουμε τα δεδομένα που θα εμφανιστούν στο Dashboard που θα παρουσιάζει τους δρόμους για του οποίους υπάρχουν μέτριες έως και χαμηλές καταγραφές σε οχήματα. Αυτό γίνεται από την `getLowCongestionData` (Listing D.10), η οποία αρχικά υπολογίζει τον συνολικό αριθμό των επιλεγμένων δεδομένων όπως και η `sortStreetData` (Listing D.8).

Η μέθοδος επιστρέφει μια υπολίστα των δεδομένων, παραλείποντας τα πρώτα στοιχεία ανάλογα με τον συνολικό αριθμό δεδομένων. Αν ο συνολικός αριθμός δεδομένων είναι μικρότερος από 20, τότε επιστρέφονται όλα τα δεδομένα. Αλλιώς, επιστρέφονται τα τελευταία 20 δεδομένα, τα οποία αντιστοιχούν στους δρόμους με τη χαμηλότερη συμφόρηση.

Listing D.10: Φιλτράρισμα δεδομένων ώστε να κρατήσουμε τις μικρότερες εγγραφές

```

1      public List<GovApiData> getLowCongestionData(List<GovApiData> ←
        streetData){
2
3      long count = streetData.stream()
4          .collect(Collectors.groupingBy(GovApiData::←
            getRoad_name))
5          .values().stream()
6          .map(roadDatas -> roadDatas.stream()
7              .max(Comparator.comparingInt(GovApiData::←
                getCountedCars)))
8          .filter(Optional::isPresent)
9          .map(Optional::get)
10         .sorted((rd1, rd2) -> rd2.getCountedCars() - rd1.←
            getCountedCars())
11         .count();
12
13     return streetData.stream()
14         .skip(count > 20 ? count - 20 : 0)
15         .collect(Collectors.toList());
16 }
```

D.1.1.5 Εγγραφή δεδομένων σε XML

Η εγγραφή δεδομένων σε αρχείο xml γίνεται από την μέθοδο `writeXML` της κλάσης `OverviewTrafficWriteToXML`. Παρακάτω θα αναλύσουμε όλα τα τμήματα του κώδικα της κλάσης αυτής που μας παρέχει το XML αρχείο.

Στο αρχικό τμήμα της κλάσης δηλώνουμε μια `private` μεταβλητή με το όνομα `OverviewTrafficWriteToXML` που θα την χρησιμοποιήσουμε στον `constructor` για να αποθηκεύσουμε τα δεδομένα κίνησης που λαμβάνουμε από τον `WebClient` (Listing D.5).

Τα δεδομένα θα ληφθούν από τη `populateLists` (Listing D.11), η οποία δέχεται μια λίστα `dataList` τύπου `GovApiData`, μια ημερομηνία έναρξης και μια ημερομηνία λήξης ως παραμέτρους. Καλεί τη μέθοδο `getGOVTrafficData` (Listing D.5) του `WebClient` για να λάβει τα δεδομένα κίνησης για το διάστημα αυτό και επιστρέφει τα αποτελέσματα. Η μέθοδος `block()` χρησιμοποιείται για να περιμένει μέχρι να ολοκληρωθεί η λήψη των δεδομένων.

Listing D.11: Δημιουργία *Constructor* και μεθόδου για την συλλογή δεδομένων στην *OverviewTrafficWriteToXML*

```

1      // Constructor
2      public OverviewTrafficWriteToXML(GovAPIService govAPIService) {
3          this.govAPIService = govAPIService;
4      }
5      // Get List Of Data
6      public List<GovApiData> populateLists(List<GovApiData> dataList, ←
          String startDate, String endDate){
7          return this.govAPIService.getGOVTrafficData(startDate, endDate←
              ).block();
8      };

```

Αφού έχουμε λάβει τα δεδομένα μπορούμε να προχωρήσουμε στην εγγραφή τους σε XML αρχείο. Αρχικά δημιουργούμε ένα νέο έγγραφο XML χρησιμοποιώντας τις κλάσεις `DocumentBuilderFactory` και `DocumentBuilder` (Listing D.12). Το έγγραφο αυτό θα περιέχει όλα τα δεδομένα κυκλοφορίας που θα εγγραφούν στο XML αρχείο. Το Document αυτό θα περιέχει το root element με όνομα “TrafficMonitoring”

Listing D.12: Δημιουργία Document για την εγγραφή του XML

```

1      DocumentBuilderFactory documentBuilderFactory = ↵
        DocumentBuilderFactory.newInstance();
2      DocumentBuilder documentBuilder = documentBuilderFactory.↵
        newDocumentBuilder();
3      Document document = documentBuilder.newDocument();
4
5
6      Element rootElement = document.createElement("TrafficMonitoring")↵
        ;
7      document.appendChild(rootElement);

```

Για κάθε δεδομένο από την λίστα που λαμβάνουμε από την `getGovTrafficData`, δημιουργούμε ένα νέο παιδί από το element “TrafficMonitoring” με το όνομα `Measurement`. Το κάθε παιδί θα περιλαμβάνει πληροφορίες όπως το `timestamp`, αισθητήρας, μέση ταχύτητα, αριθμός οχημάτων, όνομα δρόμου και πληροφορίες δρόμου. Όταν ολοκληρώνεται η δημιουργία ενός `Measurement` γίνεται `append` στο root element (Listing D.13).

Listing D.13: Προσθήκη δεδομένων στο Measurement

```

1  for(GovApiData data : dataList){
2      // Create Parent Node for each measurement
3      Element measurementElement = document.createElement("↵
        Measurement");
4      measurementElement.setAttribute("timestamp", data.↵
        getTimestamp().toString());
5
6      // Write Sensor info Average Speed //
7      Element sensorID = document.createElement("sensor");
8      sensorID.appendChild(document.createTextNode(data.↵
        getSensor()));
9      measurementElement.appendChild(sensorID);
10
11     // Write to XML Average Speed //
12     Element averageSpeed = document.createElement("↵
        averageSpeed");
13     averageSpeed.appendChild(document.createTextNode(data.↵
        getAverageSpeed().toString()));

```

```

14         measurementElement.appendChild(averageSpeed);
15
16         // Write to XML Counted Cars //
17         Element countedCars = document.createElement("countedCars↵
18         ");
19         countedCars.appendChild(document.createTextNode(data.↵
20         getCountedCars().toString()));
21         measurementElement.appendChild(countedCars);
22
23         // Write to XML Street Name //
24         Element streetName = document.createElement("roadName");
25         streetName.appendChild(document.createTextNode(data.↵
26         getRoad_name()));
27         measurementElement.appendChild(streetName);
28
29         // Write to XML roadInfo //
30         Element roadInfo = document.createElement("roadInfo");
31         roadInfo.appendChild(document.createTextNode(data.↵
32         getRoadInfo()));
33         measurementElement.appendChild(roadInfo);
34
35         rootElement.appendChild(measurementElement);
36     }

```

Ο Transformer, διαμορφώνει το XML έγγραφο και το εγγράφει σε ένα αρχείο με το όνομα "Traffic.xml" (Listing D.14). Επίσης, ορίζει τον χαρακτήρα κωδικοποίησης ως UTF-8 και ενεργοποιεί την ευαίσθητη στο διάστημα γραφή (indentation).

Listing D.14: *Εγγραφή XML αρχείου*

```

1     try (FileOutputStream output = new FileOutputStream("Traffic.↵
2     xml")) {
3         TransformerFactory transformerFactory = ↵
4         TransformerFactory.newInstance();
5         Transformer transformer = transformerFactory.↵
6         newTransformer();
7         transformer.setOutputProperty(OutputKeys.ENCODING, "UTF-8↵
8         ");
9         transformer.setOutputProperty(OutputKeys.INDENT, "yes");

```

```

6         DOMSource source = new DOMSource(document);
7         StreamResult result = new StreamResult(output);
8
9         transformer.transform(source, result);
10    } catch (IOException e) {
11        e.printStackTrace();
12    } catch (TransformerConfigurationException e) {
13        throw new RuntimeException(e);
14    } catch (TransformerException e) {
15        throw new RuntimeException(e);
16    }

```

D.1.1.6 Εγγραφή δεδομένων σε JSON

Αντίστοιχη λογική στον τρόπο που συλλέγουμε και γράφουμε το JSON αρχείο ακολουθεί και η κλάση OverviewTrafficWriteToJSON. Αφού λάβουμε τα δεδομένα μας από τον WebClient (Listing D.5) σε μια λίστα από αντικείμενα GovApiData, μπορούμε να καλέσουμε την μέθοδο writeToJSON (Listing D.15) για να μας δημιουργήσει τα JSON αντικείμενα και το αρχείο που θα λάβει ο χρήστης. Η μέθοδος αυτή δημιουργεί ένα JSONArray από τη λίστα δεδομένων και γράφει το JSONArray σε ένα αρχείο JSON. Αν υπάρξει κάποιο σφάλμα κατά τη διάρκεια της διαδικασίας, θα εκτυπωθεί η αντίστοιχη στοιβή κλήσεων.

Listing D.15: *Εγγραφή σε αρχείο JSON*

```

1 public class OverviewTrafficWriteToJSON {
2     private final GovAPIService govAPIService;
3     public OverviewTrafficWriteToJSON(GovAPIService govAPIService) {
4         this.govAPIService = govAPIService;
5     }
6
7     public List<GovApiData> populateLists(List<GovApiData> dataList, ↵
        String startDate, String endDate) {
8         return this.govAPIService.getGOVTrafficData(startDate, endDate ↵
            ).block();
9     }
10
11    public void writeToJSON(List<GovApiData> dataList) {

```

```
12     JSONArray jsonArray = new JSONArray(dataList);
13     String filePath = "file.json";
14     try (FileWriter fileWriter = new FileWriter(filePath)) {
15         fileWriter.write(jsonArray.toString());
16         System.out.println("JSON written to " + filePath);
17     } catch (IOException e) {
18         e.printStackTrace();
19     }
20 }
21 }
```

D.1.1.7 Εγγραφή δεδομένων σε Excel

Η εγγραφή του υπολογιστικού φύλλου γίνεται από την μέθοδο `writeXLSX` (Listing D.16) που δέχεται σαν παράμετρο μια λίστα `dataList` τύπου `GovApiData`. Σαν πρώτο βήμα δημιουργείτε ένα νέο αρχείο XLSX και ένα φύλλο εργασίας με το όνομα “Athens Traffic Report” που θα παρουσιάσουν όλα τα δεδομένα. Έπειτα όπως αναφέραμε και σε προηγούμενο κεφάλαιο σημαντικό είναι τα δεδομένα μας να παρουσιάζονται με ένα ευανάγνωστο τρόπο προς τον χρήστη, να παρουσιαστούν δηλαδή σε μια μορφή πίνακα. Το συγκεκριμένο εγχείρημα το επιτυγχάνουμε με την δημιουργία δύο διαφορετικών styles. Πιο αναλυτικά τα styles αυτά έχουν τις παρακάτω ιδιότητες

- **Header Style:** εφαρμόζεται στα κελιά της επικεφαλίδας. Ορίζει ένα παχύ περίγραμμα για τα κελιά και χρησιμοποιεί ένα γραμματοσειρά που είναι έντονη (bold) και με μέγεθος 14.
- **Cell Style:** εφαρμόζεται στα υπόλοιπα κελιά του φύλλου εργασίας. Ορίζει ένα λεπτό περίγραμμα για τα κελιά.

Όπως θα μελετήσουμε και στην συνέχεια κάθε κελί που δημιουργείται, εφαρμόζει το style (Listing D.16). Αυτό σημαίνει ότι κάθε κελί θα έχει ένα λεπτό περίγραμμα γύρω του. Το `headerStyle` εφαρμόζεται μόνο στα κελιά της επικεφαλίδας, δίνοντας τους ένα πιο έντονο στυλ σε σύγκριση με τα υπόλοιπα κελιά.

Listing D.16: Δημιουργία *workbook* και *style* των κελιών

```

1 public void writeXLSX(List<GovApiData> dataList){
2     XSSFWorkbook workbook = new XSSFWorkbook();
3     XSSFSheet spreadsheet = workbook.createSheet(" Athens Traffic↔
        Report ");
4
5
6     XSSFCellStyle headerStyle = workbook.createCellStyle();
7     headerStyle.setBorderTop(BorderStyle.THICK);
8     headerStyle.setBorderBottom(BorderStyle.THICK);
9     headerStyle.setBorderLeft(BorderStyle.THICK);
10    headerStyle.setBorderRight(BorderStyle.THICK);
11
12    XSSFFont font = workbook.createFont();
13    font.setFontHeightInPoints((short)14);
14    font.setBold(true);
15
16    headerStyle.setFont(font);
17
18    XSSFCellStyle style = workbook.createCellStyle();
19    style.setBorderTop(BorderStyle.THIN);
20    style.setBorderBottom(BorderStyle.THIN);
21    style.setBorderLeft(BorderStyle.THIN);
22    style.setBorderRight(BorderStyle.THIN);
23
24    ....
25 }
```

Έπειτα δημιουργείτε η πρώτη γραμμή του υπολογιστικού φύλλου στην θέση μηδέν όπου θα μπουν και οι επικεφαλίδες του πίνακα. Με την `setCellValue` ορίζουμε το όνομα που θα έχει η επικεφαλίδα καθώς και στο τέλος καθορίζουμε και το `style` του. Αυτή η διαδικασία επαναλαμβάνεται για κάθε κελί της επικεφαλίδας, δημιουργώντας ένα κελί για κάθε στήλη του φύλλου εργασίας με τα ονόματα “Timestamp”, “Sensor ID”, “Road Name”, “Counted Cars”, “Average Speed”, “Road Info”(Listing D.17).

Listing D.17: Δημιουργία *Header* υπολογιστικού φύλλου

1

```

2      Row header = spreadsheet.createRow(0);
3
4      Cell headerCell = header.createCell(0);
5      headerCell.setCellValue("Timestamp");
6      headerCell.setCellStyle(headerStyle);
7
8      headerCell = header.createCell(1);
9      headerCell.setCellValue("Sensor ID");
10     headerCell.setCellStyle(headerStyle);
11
12     headerCell = header.createCell(2);
13     headerCell.setCellValue("Road Name");
14     headerCell.setCellStyle(headerStyle);
15
16     headerCell = header.createCell(3);
17     headerCell.setCellValue("Counted Cars");
18     headerCell.setCellStyle(headerStyle);
19
20     headerCell = header.createCell(4);
21     headerCell.setCellValue("Average Speed");
22     headerCell.setCellStyle(headerStyle);
23
24     headerCell = header.createCell(5);
25     headerCell.setCellValue("Road Info");
26     headerCell.setCellStyle(headerStyle);

```

Συνεχίζοντας στην εγγραφή των δεδομένων στα κελιά ο κώδικας δημιουργεί μια νέα γραμμή για κάθε στοιχείο της λίστας `dataList` και συμπληρώνει τα κελιά αυτής της γραμμής με τα δεδομένα κίνησης (Listing D.18). Αρχικά αρχικοποιεί έναν μετρητή `cellCounter` για την παρακολούθηση του αριθμού της γραμμής που δημιουργείται. Έπειτα συνεχίζει με την εγγραφή των δεδομένων μέσα από την λίστα `dataList`. Στην συγκεκριμένη διαδικασία για κάθε `GovApiData` αντικείμενο ακολουθούνται τα εξής βήματα.

1. Αυξάνει τον μετρητή κατά ένα για κάθε νέο στοιχείο της λίστας (`cellCounter++`).
2. Δημιουργεί μια νέα γραμμή στο φύλλο εργασίας στη θέση που καθορίζεται από τον μετρητή (`Row row = spreadsheet.createRow(cellCounter)`).

3. Δημιουργεί ένα νέο κελί στη γραμμή (`Cell cell = row.createCell(0)`)
4. Ορίζει την τιμή του κελιού (`cell.setCellValue(data.getTimestamp().toString())`).
5. Εφαρμόζει το στυλ `style` στο κελί(`cell.setCellStyle(style)`).

Listing D.18: Προσθήκη δεδομένων στα κελιά

```
1      ....
2      int cellCounter = 0;
3
4      for(GovApiData data : dataList){
5          cellCounter++;
6          Row row = spreadsheet.createRow(cellCounter);
7
8          Cell cell = row.createCell(0);
9          cell.setCellValue(data.getTimestamp().toString());
10         cell.setCellStyle(style);
11
12         cell = row.createCell(1);
13         cell.setCellValue(data.getSensor());
14         cell.setCellStyle(style);
15
16         cell = row.createCell(2);
17         cell.setCellValue(data.getRoad_name());
18         cell.setCellStyle(style);
19
20         cell = row.createCell(3);
21         cell.setCellValue(data.getCountedCars());
22         cell.setCellStyle(style);
23
24         cell = row.createCell(4);
25         cell.setCellValue(data.getAverageSpeed());
26         cell.setCellStyle(style);
27
28         cell = row.createCell(5);
29         cell.setCellValue(data.getRoadInfo());
30         cell.setCellStyle(style);
31     }
```

Αφού γράψουμε τα δεδομένα στα κελιά, ο κώδικας θα αναπροσαρμόσει το πλάτος των στηλών του φύλλου εργασίας Excel για να ταιριάζει με το περιεχόμενο. Έπειτα ακολουθεί και η εγγραφή του σε `xlsx` αρχείο μέσα από `InputStream` (Listing D.19)

Listing D.19: *Εγγραφή αρχείου XLSX*

```
1
2     spreadsheet.autoSizeColumn(0);
3     spreadsheet.autoSizeColumn(1);
4     spreadsheet.autoSizeColumn(2);
5     spreadsheet.autoSizeColumn(3);
6     spreadsheet.autoSizeColumn(4);
7     spreadsheet.autoSizeColumn(5);
8
9     try {
10         FileOutputStream out = new FileOutputStream("Traffic ↔
11             Report.xlsx");
12         workbook.write(out);
13         out.close();
14         workbook.close();
15         System.out.println("Excel written successfully ..");
16
17     } catch (FileNotFoundException e) {
18         e.printStackTrace();
19     } catch (IOException e) {
20         e.printStackTrace();
21     }
```

D.1.1.8 Σχεδιασμός API

Η επικοινωνία μεταξύ του Frontend και του Backend πραγματοποιείται μέσα από διάφορα REST API. Το REST API είναι ένα σύνολο κανόνων και συμβάσεων που επιτρέπουν την επικοινωνία αυτή. Πιο αναλυτικά η δημιουργία ενός τέτοιου Rest API στο Spring Boot γίνεται μέσα από τους Controllers όπου διαχειρίζονται και τις αιτήσεις HTTP. Κάθε Controller είναι μια κλάση που επισημαίνεται με την ετικέτα `@RestController`. Οι μέθοδοι μέσα σε έναν Controller, γνωστές ως `handler methods`, επισημαίνονται με ετικέτες όπως `@RequestMapping`

ping ή @GetMapping και αντιστοιχούν σε συγκεκριμένες διαδρομές URL. Αυτές οι μέθοδοι χειρίζονται τη λογική επεξεργασίας για τις αντίστοιχες αιτήσεις, όπως η λήψη και επεξεργασία των δεδομένων από το εξωτερικό API του gov ή την δημιουργία και αποστολή στο frontend τα xml, json και Excel αρχεία. Με αυτό τον τρόπο, οι Controllers στο Spring Boot είναι ζωτικής σημασίας για την επικοινωνία μεταξύ του backend και του frontend.

Για την αποστολή όλων των δεδομένων της κυκλοφοριακής κατάστασης χρησιμοποιείται ο GOVApiController (Listing D.20). Αρχικά θέτουμε τα απαραίτητα annotation για να δηλώσουμε ότι ο παρακάτω κώδικας είναι ένα Rest API RestController, RequestMapping, CrossOrigin. Η ετικέτα (*) στο CrossOrigin μας δηλώνει ότι επιτρέπονται οι Cross-Origin αιτήσεις από όλα τα domain. Σαν υλοποίηση δηλώνουμε ότι η μέθοδος που περιέχει ο Controller μέσα θα χειρίζεται τις αιτήσεις GET για το URL “/gov”. Προϋπόθεση είναι στην HTTP κλήση να δεχτεί δύο παραμέτρους startDate και endDate και στην συνέχεια να καλέσει τον WebClient (Listing D.5) που βρίσκεται στο Business Layer και να επιστρέψει ένα Mono που περιέχει μια λίστα αντικειμένων GovApiData.

Listing D.20: *Controller για την λήψη όλων των δεδομένων κυκλοφορίας*

```

1      @RestController
2      @RequestMapping
3      @CrossOrigin("*")
4      public class GOVApiController {
5          private final GovAPIService govAPIService;
6          public GOVApiController(GovAPIService govAPIService) {
7              this.govAPIService = govAPIService;
8          }
9
10         @GetMapping(value = "/gov")
11         public Mono<List<GovApiData>> fetchData(@RequestParam String ←
            startDate, @RequestParam String endDate) throws ParseException ←
            {
12             startDate = ConvertDateFormat.changeDateFormat(startDate);
13             endDate = ConvertDateFormat.changeDateFormat(endDate);
14             System.out.println(startDate + " — " + endDate);
15             return govAPIService.getGOVTrafficData(startDate, endDate);
16         }
17     }

```

Με αντίστοιχο τρόπο υλοποιούμε και τους υπόλοιπους Controllers. Για την αποστολή των αρχείων XML,JSON και Excel υπεύθυνος είναι ο OverviewTrafficXMLController. Μέσα σε αυτόν τον Controller δημιουργούμε 3 διαφορετικά endpoints που θα χειρίζονται τις αιτήσεις GET για τα URL “/getXML”, “/getXLSX” και “/getJSON” αντίστοιχα. Σαν παραμέτρους αυτά τα Rest API δέχονται δύο παραμέτρους, startDate και endDate, και δημιουργούν αντίστοιχα ένα αρχείο XML(Listing D.21), XLSX(Listing D.23) και JSON(Listing D.22) με τα δεδομένα κίνησης για το διάστημα αυτό. Τα αρχεία αυτά επιστρέφονται στο frontend ως λήψεις αρχείων.

Listing D.21: *Endpoint αποστολή του XML αρχείου*

```

1  @GetMapping(value = "/getXML")
2  public void getXMLFile(HttpServletRequest response, @RequestParam ↵
    String startDate, @RequestParam String endDate) throws ↵
    IOException, FileNotFoundException, ↵
    ParserConfigurationException, ParseException {
3      List<GovApiData> dataList = new ArrayList<>();
4      startDate = ConvertDateFormat.changeDateFormat(startDate);
5      endDate = ConvertDateFormat.changeDateFormat(endDate);
6      OverviewTrafficWriteToXML overviewTrafficWriteToXML = new ↵
        OverviewTrafficWriteToXML(govAPIService);
7      dataList = overviewTrafficWriteToXML.populateLists(dataList, ↵
        startDate, endDate);
8      overviewTrafficWriteToXML.writeXML(dataList);
9      output = new File("Traffic.xml");
10     try(InputStream inputStream = new FileInputStream(output)){
11         response.setContentType("application/force-download");
12         response.setHeader("Content-Disposition", "attachment; ↵
            filename="+ "Athens-Traffic.xml");
13         IOUtils.copy(inputStream, response.getOutputStream());
14         response.flushBuffer();
15     }
16 }
17 }

```

Listing D.22: *Endpoint αποστολή του JSON αρχείου*

```

1  @GetMapping(value = "/getJSON")
2  public void getJSONFile(HttpServletResponse response,↵
    @RequestParam String startDate, @RequestParam String endDate) ↵
    throws IOException, ParseException {
3      List<GovApiData> dataList = new ArrayList<>();
4      startDate = ConvertDateFormat.changeDateFormat(startDate);
5      endDate = ConvertDateFormat.changeDateFormat(endDate);
6      OverviewTrafficWriteToJSON overviewTrafficWriteToJSON = new ↵
        OverviewTrafficWriteToJSON(govAPIService);
7      dataList = overviewTrafficWriteToJSON.populateLists(dataList,↵
        startDate, endDate);
8      overviewTrafficWriteToJSON.writeToJSON(dataList);
9      output = new File("file.json");
10
11     try(InputStream inputStream = new FileInputStream(output)){
12         response.setContentType("application/force-download");
13         response.setHeader("Content-Disposition", "attachment; ↵
            filename="+ "file.json");
14         IOUtils.copy(inputStream, response.getOutputStream());
15         response.flushBuffer();
16     } catch (FileNotFoundException e) {
17         throw new RuntimeException(e);
18     }
19 }

```

Listing D.23: *Endpoint αποστολή του Excel αρχείου*

```

1  @GetMapping(value = "/getXLSX")
2  public void getXLSXFile(HttpServletResponse response,↵
    @RequestParam String startDate, @RequestParam String endDate)↵
    throws IOException, ParseException {
3      List<GovApiData> dataList = new ArrayList<>();
4      startDate = ConvertDateFormat.changeDateFormat(startDate);
5      endDate = ConvertDateFormat.changeDateFormat(endDate);
6      OverviewTrafficWriteToXLSX overviewTrafficWriteToXLSX = new ↵
        OverviewTrafficWriteToXLSX(govAPIService);
7      dataList = overviewTrafficWriteToXLSX.populateLists(dataList,↵

```

```

        startDate, endDate);
8      overviewTrafficWriteToXLSX.writeXLSX(dataList);
9      output = new File("Traffic-Report.xlsx");
10
11     try(InputStream inputStream = new FileInputStream(output)){
12         response.setContentType("application/force-download");
13         response.setHeader("Content-Disposition", "attachment; ↵
            filename="+ "Traffic-Report.xlsx");
14         IOUtils.copy(inputStream, response.getOutputStream());
15         response.flushBuffer();
16     } catch (FileNotFoundException e) {
17         throw new RuntimeException(e);
18     }
19 }

```

Για τα δεδομένα που προβάλλονται στο Dashboard χρησιμοποιήθηκαν διαφορετικός Controller. Ο Controller αυτός περιλαμβάνει υλοποιήσεις από Rest API που θα επιστρέψουν στο frontend τα δεδομένα που θα προβληθούν στα τέσσερα διαδοχικά πάνελ που θα περιλαμβάνουν τις γενικές πληροφορίες για την κατάσταση στο οδικό δίκτυο, το περιεχόμενο που θα οπτικοποιηθεί σε διαγράμματα για τους δρόμους με την μεγαλύτερη κίνηση και τα δεδομένα που θα προβληθούν σε διαγράμματα για την μέτρια έως χαμηλή κίνηση. Κάθε ένα από αυτά αντιστοιχίζεται σε διαφορετικό GET endpoint τα οποία θα αναλύσουμε παρακάτω.

Την αποστολή των γενικών πληροφοριών αναλαμβάνει η μέθοδος miniCardData (Listing D.24). Το endpoint που αντιστοιχεί στην συγκεκριμένη μέθοδο είναι το /getMiniCardData. Η συγκεκριμένη μέθοδος δέχεται σαν παραμέτρους το startDate και το endDate και αφού τα προσαρμόσει στο απαραίτητο format θα τα περάσει στον WebClient (Listing D.5) ώστε να λάβουμε τα δεδομένα για εκείνη την χρονική περίοδο. Έχοντας πλέον τα δεδομένα μας σε λίστα θα εξάγουμε τις γενικές πληροφορίες κίνησης από την μέθοδο getMiniCardData (Listing D.7) της κλάσης DashboardFeatures. Η συγκεκριμένη μέθοδος θα μας επιστέψει μια λίστα τύπου Long με τα δεδομένα που θα προβληθούν. Πιο αναλυτικά στην πρώτη θέση του πίνακα θα περιέχει τον συνολικό αριθμό οχημάτων που καταγράφηκαν στο οδικό δίκτυο της Αθήνας για την συγκεκριμένη χρονική περίοδο. Στην δεύτερη θέση του πίνακα θα περιλαμβάνει την μέση ταχύτητα των οχημάτων, στην τρίτη θέση τον συνολικό αριθμό των δρόμων που υπάρχουν καταγραφές και τέλος στην τέταρτη θέση θα περιέχει τον αριθμό των

διαθέσιμων αισθητήρων.

Listing D.24: *Endpoint για τα δεδομένα στα mini cards*

```

1  @GetMapping(value = "/getMiniCardData")
2  public List<Long> miniCardData(@RequestParam String startDate,
3                                @RequestParam String endDate) ←
4                                throws ParseException {
5      startDate = ConvertDateFormat.changeDateFormat(startDate);
6      endDate = ConvertDateFormat.changeDateFormat(endDate);
7      DashboardFeatures dashboardFeatures = new DashboardFeatures(←
8          this.govAPIService);
9      return dashboardFeatures.getMiniCardData(startDate, endDate);
10 }

```

Οι μέθοδοι που επιστρέφουν το περιεχόμενο που θα προβληθεί από τα διαγράμματα, ακολουθούν παρόμοια λογική. Αρχικά μέσα στον Controller έχει υλοποιηθεί μια μέθοδος η οποία θα μας κάνει ταξινόμηση τα δεδομένα που λαμβάνουμε από τον WebClient (Listing D.5). Την λίστα αυτή χρησιμοποιούν οι μέθοδοι `getTenStreetsWithHighestCarCount` (Listing D.9) και `getLowCongestionData` (Listing D.10) για να καλέσουν τις απαραίτητες java κλάσεις από το business layer που θα τους επιστρέψει την πληροφορία που θα προβληθεί στο frontend.

Listing D.25: *Ταξινόμηση δεδομένων από τον WebClient*

```

1  private void sortData(String startDate, String endDate) throws ←
2      ParseException {
3      startDate = ConvertDateFormat.changeDateFormat(startDate);
4      endDate = ConvertDateFormat.changeDateFormat(endDate);
5      DashboardFeatures dashboardFeatures = new DashboardFeatures(←
6          this.govAPIService);
7      sortedData = new ArrayList<>(dashboardFeatures.sortStreetData(←
8          (startDate, endDate)));
9  }

```

Για τους δρόμους με την μεγαλύτερη προέλευση οχημάτων δημιουργήθηκε το `/getTenHighest-CarCountStreets` endpoint. Στο οποίο αφού πάρουμε τα ταξινομημένα δεδομένα θα καλέσουμε την μέθοδο `getTenStreetsWithHighestCarCount` (Listing D.9) από το business layer που θα μας επιστρέψει μια λίστα από αντικείμενα με τους δέκα δρόμους με την μεγαλύτερη προέλευση οχημάτων, την οποία θα λάβει το frontend.

Listing D.26: Αποστολή δεδομένων για την μεγαλύτερη προέλευση οχημάτων

```

1  @GetMapping(value = "/getTenHighestCarCountStreets")
2  public List<GovApiData> tenHighestCarCount(@RequestParam String ↵
    startDate,
3
    @RequestParam String ↵
    endDate) throws ↵
    ParseException {
4
    sortData(startDate, endDate);
5    DashboardFeatures dashboardFeatures = new DashboardFeatures(↵
        this.govAPIService);
6    List<GovApiData> highCongestionData = sortedData;
7    return dashboardFeatures.getTenStreetsWithHighestCarCount(↵
        highCongestionData);
8 }

```

Για τους δρόμους με μέτρια έως και χαμηλή προέλευση οχημάτων ακολουθούμε παρόμοια λογική. Αρχικά λαμβάνουμε τα δεδομένα που ταξινομήθηκαν και στην συνέχεια παίρνουμε το αποτέλεσμα της μεθόδου `getLowCongestionData` (Listing D.10) που περιέχει τους είκοσι δρόμους που θα προβληθούν στο Dashboard για το “Medium/Low Congestion”

Listing D.27: Αποστολή δεδομένων για την μεσαία χαμηλή προέλευση οχημάτων

```

1  @GetMapping(value = "/getLowCongestionData")
2  public List<GovApiData> lowCongestionStreets(@RequestParam String↵
    startDate,
3
    @RequestParam String↵
    endDate) throws↵
    ParseException ↵
    {
4
    sortData(startDate, endDate);
5    DashboardFeatures dashboardFeatures = new DashboardFeatures(↵
        this.govAPIService);
6    return dashboardFeatures.getLowCongestionData(sortedData);
7 }

```

D.2 Frontend

Το frontend αποτελεί το μέρος της εφαρμογής με το οποίο αλληλοεπιδρά ο χρήστης. Είναι η εφαρμογή που θα τρέχει στον περιηγητή και θα περιέχει διάφορα γραφικά στοιχεία μέσω των οποίων ο χρήστης θα επικοινωνεί με την εφαρμογή.

D.2.1 Angular και Angular Material

Για την ανάπτυξη της frontend εφαρμογής χρησιμοποιήθηκε η Angular σε συνδυασμό με το Material Library. Σαν βασικό framework η Angular προσφέρει μια σταθερή και αποδοτική βάση για την δημιουργία σύνθετων εφαρμογών. Από την άλλη πλευρά το Material Library μας προσφέρει μια σειρά από επαναχρησιμοποιούμενα γραφικά στοιχεία που μπορούμε να χρησιμοποιήσουμε, όπως πίνακες, ημερολόγια, κάρτες για την παρουσίαση δεδομένων και πολλά άλλα. Τα συγκεκριμένα γραφικά στοιχεία έρχονται με προκαθορισμένο design, ωστόσο μπορούμε να εύκολο να προσαρμόσουμε την εμφάνιση τους μέσα από CSS.

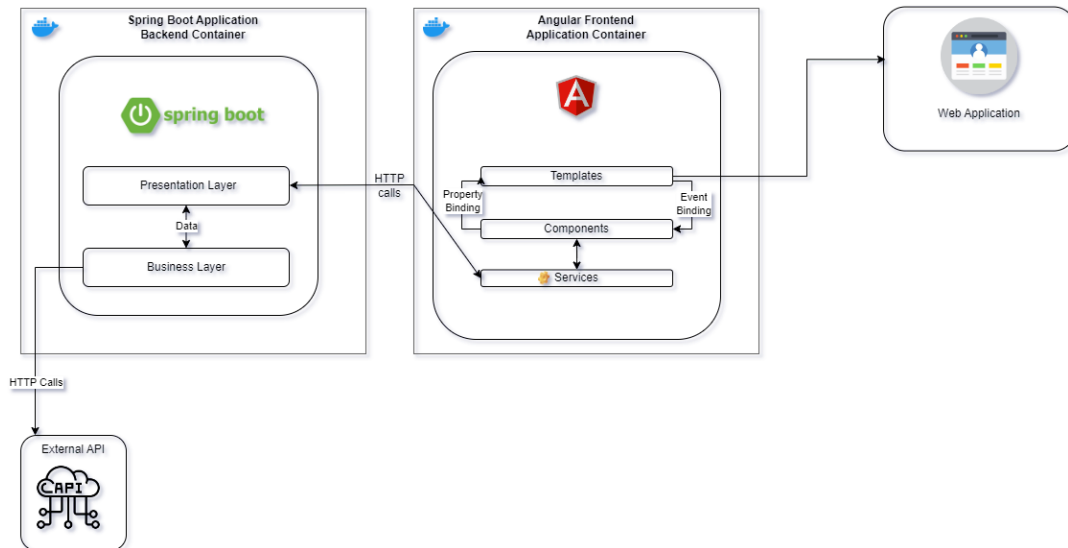
Για την οπτικοποίηση των δεδομένων χρησιμοποιήθηκε το ChartJS. Το ChartJS είναι μια δωρεάν βιβλιοθήκη JavaScript για την δημιουργία διαγραμμάτων βασισμένα σε HTML. Ως βιβλιοθήκη μας προσφέρει πολλούς ενσωματωμένους τύπους διαγραμμάτων για να παρουσιάσουμε τα δεδομένα.

D.2.1.1 Αρχιτεκτονική Frontend

Η αρχιτεκτονική της εφαρμογής αποτελείται από κάποια βασικά τμήματα. Κάθε ένα από αυτά διαδραματίζει σημαντικό ρόλο στην εύρυθμη λειτουργία της εφαρμογής. Τα τμήματα αυτά αποτελούνται από τα services, τα components και τα templates. Με βάση το παρακάτω διάγραμμα θα περιγράψουμε την αρχιτεκτονική της εφαρμογής από τα κάτω επίπεδα προς τα πάνω.

Ως πιο χαμηλό επίπεδο στην αρχιτεκτονική της frontend εφαρμογής (Σχήμα D.24) καθορίζουμε τα services ή αλλιώς υπηρεσίες. Στις συγκεκριμένες υπηρεσίες τοποθετούμε την λογική η οποία μπορεί να χρησιμοποιηθεί από διαφορετικά components. Στην εφαρμογή ο ρόλος των services είναι να λαμβάνουν και να αποστέλλουν HTTP κλήσεις με δεδομένα προς τους Controllers του Backend που βρίσκονται όπως εξετάσαμε και προηγουμένως στο Presentation Layer.

Τα δεδομένα που ανταλλάσσονται από τα services μπορούν να τα χρησιμοποιήσουν τα com-



Σχήμα D.24: Αρχιτεκτονική Angular Frontend

ponents. Σκοπός τους είναι να επεξεργάζονται τα δεδομένα αυτά και να παρέχουν μια λειτουργικότητα, καθώς και μέσα από την TypeScript να καθορίσουν την συμπεριφορά των γραφικών στοιχείων. Πιο αναλυτικά σε αντίθεση με τα services, τα components μπορούν και να ελέγχουν το περιεχόμενο που προβάλλεται στην οθόνη

Το περιεχόμενο που προβάλλεται στη οθόνη καθορίζεται από την HTML και CSS του κάθε component. Τα συγκεκριμένα αρχεία λαμβάνουν τα δεδομένα και την λογική από την TypeScript που υπάρχει στα components και είναι υπεύθυνα μόνο για την προβολή τους.

Ο διαχωρισμός για τον αριθμό των Components και Services αποτελεί επιλογή του σχεδιαστή καθώς και μπορεί να επηρεαστεί από την πολυπλοκότητα του συστήματος που θέλουμε να αναπτύξουμε.

D.2.1.2 Σχεδιασμός Services

Το Service που είναι υπεύθυνο για να την λήψη των δεδομένων που θα χρησιμοποιηθούν για το Dashboard από το Backend είναι το DashboardService (Listing D.28). Με το Inject δηλώνουμε ότι το Service αυτό μπορεί να εισαχθεί και να χρησιμοποιηθεί από οποιοδήποτε component. Αρχικά ορίζουμε τα διαφορετικά endpoint του API που θα κάνουμε τις GET

HTTP κλήσεις, αυτά είναι :

1. Backend API endpoint: /getTenHighestCarCountStreets (Listing D.26)
2. Backend API endpoint: /getLowCongestionData (Listing D.27)
3. Backend API endpoint: /getMiniCardData (Listing D.24)

Οι μέθοδοι `getChartData` και `getLowCongestionData` λειτουργούν με παρόμοιο τρόπο. Και οι δύο δέχονται παραμέτρους οι οποίες θα περιλαμβάνουν το `Start Date` και `End date` που διάλεξε ο χρήστης και θα κάνουν μια GET κλήση στο αντίστοιχο endpoint του API. Η απάντηση που θα λάβουν πίσω καθορίζεται από μια διεπαφή, `TableItem` που ορίζει τη δομή των αντικειμένων που επιστρέφονται από τα endpoints.

Παρόμοια λογική ακολουθεί και η `getMiniCardData`, για την όποια θα γίνει κλήση στο αντίστοιχο endpoint του API περνώντας στο HTTP request τις απαραίτητες παραμέτρους και θα λάβει μια απάντηση. Ειδοποιός διαφορά μεταξύ της `getMiniCardData` και των μεθόδων που περιγράφηκαν προηγουμένως είναι ο τύπος απάντησης που περιμένει, ο οποίος είναι μια λίστα από αριθμούς.

Listing D.28: *Dashboard Service για την επικοινωνία με το Backend*

```

1  @Injectable({
2    providedIn: 'root'
3  })
4  export class DashboardService {
5    private apiUrl = 'http://' + HOST + ':' + PORT + '/' +
      'getTenHighestCarCountStreets' ;
6    private lowCongestionUrl = 'http://' + HOST + ':' + PORT + '/' +
      'getLowCongestionData' ;
7    private miniCardUrl = 'http://' + HOST + ':' + PORT + '/' +
      'getMiniCardData' ;
8    constructor(private http: HttpClient) { }
9
10   getChartData(requestParams : HttpParams): Observable<TableItem[]> {
11     return this.http.get<TableItem[]>(this.apiUrl, {params :
      requestParams});
12   }
13
14
```

```

15  getMiniCardData(requestParams : HttpParams): Observable<number[]> {
16      return this.http.get<number[]>(this.miniCardUrl, {params : ↵
          requestParams});
17  }
18
19  getLowCongestionData(requestParams : HttpParams): Observable<↵
      TableItem[]> {
20      return this.http.get<TableItem[]>(this.lowCongestionUrl, {params ↵
          : requestParams});
21  }
22 }
23
24 export interface TableItem {
25     road_name: string;
26     road_info: string;
27     appprocesstime: string;
28     countedcars: number;
29     average_speed: number;
30 }

```

Για την λήψη των δεδομένων του πίνακα και των αρχείων από το Data Overview χρησιμοποιείται διαφορετικό service. Σε αυτό το Service καλούμε τα εξής endpoints του API

1. Backend API endpoint: /gov (Listing D.20)
2. Backend API endpoint: /getXML (Listing D.21)
3. Backend API endpoint: /getJSON (Listing D.22)
4. Backend API endpoint: /getXLSX (Listing D.23)

Οι μέθοδοι που περιλαμβάνει είναι οι fetchTableData και downloadTableData (Listing D.29). Και οι δύο αυτοί μέθοδοι κάνουν μια GET κλήση στο Backend με παραμέτρους την χρονική περίοδο που έχει επιλέξει ο χρήστης μέσα από το UI. Ωστόσο περιμένουν διαφορετικές απαντήσεις. Η fetchTableData περιμένει μια απάντηση που θα αποτελείται από μια λίστα η οποία θα περιέχει αντικείμενα της διεπαφής TableItem, ενώ η downloadTableData αναμένει ένα αρχείο, το οποίο αποτελεί και αυτό που θα κατεβεί από τον περιηγητή του χρήστη.

Listing D.29: *Table Service για την επικοινωνία με το Backend*

```

1  export class TableService {
2
3      private apiUrl = 'http://' + HOST + ':' + PORT + '/gov' ;
4      private getXMLfile = 'http://' + HOST + ':' + PORT + '/getXML';
5      private getXLSXfile = 'http://' + HOST + ':' + PORT + '/getXLSX';
6      private getJSONfile = 'http://' + HOST + ':' + PORT + '/getJSON';
7      downloadUrl = '';
8
9      constructor(private http: HttpClient) { }
10
11     fetchTableData(requestParams : HttpParams): Observable<TableItem[]>↵
12         {
13             return this.http.get<TableItem[]>(this.apiUrl, {params : ↵
14                 requestParams});
15         }
16
17     downloadTableData(fileType: string, requestParams : HttpParams): ↵
18         Observable<Blob> {
19         if (fileType === 'xml') {
20             this.downloadUrl = this.getXMLfile;
21             console.log(this.downloadUrl)
22         }else if(fileType === 'xlsx'){
23             this.downloadUrl = this.getXLSXfile;
24         }else if(fileType === 'json'){
25             this.downloadUrl = this.getJSONfile;
26         }
27         return this.http.get(`${this.downloadUrl}`, {responseType: 'blob'↵
28             ,params: requestParams})
29     }
30
31     export interface TableItem {
32         road_name: string;
33         road_info: string;
34         appprocesstime: string;
35         countedcars: number;
36         average_speed: number;
37     }
38 }

```

D.2.1.3 Σχεδιασμός Components

Στην εφαρμογή που αναπτύξαμε, δημιουργήσαμε επτά components. Πιο αναλυτικά ο ρόλος και η λειτουργία του κάθε component φαίνεται στον παρακάτω πίνακα με μια περιληπτική περιγραφή. Στο παρακάτω υποκεφάλαιο θα μελετήσουμε τις πιο βασικές λειτουργίες των components αυτών.

Πίνακας D.1: *Components εφαρμογής*

Components	Περιγραφή
DashBoardCompoent	Περιλαμβάνει όλη την λογική για το Dashboard, επεξεργάζεται τις ημερομηνίες που δίνει ο χρήστης, παίρνει δεδομένα από το DashboardService και καλεί τα PieCompoent, SpeedLineCompoent, StatusTable component για τις οπτικοποιήσεις.
TableComponent	Περιλαμβάνει όλη την λογική για το Data Overview, παίρνει δεδομένα από το TableService.
NavigationComponent	Καθορίζει την πλοήγηση μεταξύ των τμημάτων της εφαρμογής
InfoCompoent	Παρέχει το API Documentation
SpeedLineCompoent	Component για την αναπαράσταση του διαγράμματος γραμμής.
StatusTableComponent	Component για την αναπαράσταση του πίνακα με τον μέσο όρο ταχυτήτων.
PieComponent	Component για την αναπαράσταση του διαγράμματος πίτας.

Την δημιουργία του πλαϊνού μενού την επιτυγχάνουμε με το sidenav-container της angular. Μέσα από αυτόν τον container χωρίζουμε το περιεχόμενο της οθόνης σε δύο τμήματα

1. Το mat-sidenav είναι το πλαϊνό μενού που περιέχει τα στοιχεία του μενού.
2. Το mat-sidenav-content περιέχει το κύριο περιεχόμενο της σελίδας.

Μέσα από το `mat-nav-list` περιέχουμε τα στοιχεία του μενού, τα οποία είναι σύνδεσμοι για ανακατεύθυνση μεταξύ των τμημάτων της εφαρμογής, όπως είναι το Dashboard και το Data Overview. Η ανακατεύθυνση αυτή γίνεται από τον δρομολογητή router. Επίσης με την `is-Handset` μπορούμε να κάνουμε το μενού δυναμικό, δηλαδή να δίνει στον χρήστη την επιλογή να κλείνει και να ανοίγει. Τέτοια λειτουργικότητα αποδεικνύεται αρκετά χρήσιμη όταν χρησιμοποιούμε την εφαρμογή από μικρότερες οθόνες όπως είναι αυτές των κινητών τηλεφώνων και tablet. Τις λειτουργίες αυτές τις αναλαμβάνει το Navigation Component (Listing D.30) που έχει δημιουργηθεί για τον σκοπό αυτό.

Listing D.30: *Navigation Componet για την δημιουργία δυναμικού μενού*

```

1 <mat-sidenav-container class="sidenav-container">
2   <mat-sidenav #drawer class="sidenav" fixedInViewport
3     [attr.role]="(isHandset$ | async) ? 'dialog' : '↔
      navigation'"
4     [mode]="(isHandset$ | async) ? 'over' : 'side'"
5     [opened]="(isHandset$ | async) === false">
6     <mat-toolbar>Menu</mat-toolbar>
7     <mat-nav-list>
8       <a mat-list-item href="/dashboard">Dashboard</a>
9       <a mat-list-item href="/table">Data Overview</a>
10    </mat-nav-list>
11
12    <div class="bottom-center">
13      <button mat-button (click)="openDialog()" color="primary" class↵
        ="info-button"><mat-icon>api</mat-icon>API Info</button>
14    </div>
15  </mat-sidenav>
16  <mat-sidenav-content>
17    <mat-toolbar color="primary">
18      <button
19        type="button"
20        aria-label="Toggle sidenav"
21        mat-icon-button
22        (click)="drawer.toggle()"
23        *ngIf="isHandset$ | async">
24      <mat-icon aria-label="Side nav toggle icon">menu</mat-icon>
25    </button>

```

```

26     <button mat-icon-button (click)="drawer.toggle()">
27         <mat-icon aria-label="Side nav toggle icon">menu</mat-icon>
28     </button>
29     <span>Traffic View</span>
30 </mat-toolbar>
31 <router-outlet></router-outlet>
32 </mat-sidenav-content>
33 </mat-sidenav-container>

```

Παρακάτω θα εξετάσουμε τον τρόπο λειτουργίας και τα βασικά κομμάτια κώδικα του Data Overview. Σε αυτό το τμήμα της εφαρμογής ο χρήστης μπορεί να επιλέξει μια ημερομηνία μέσα από ένα αναδυόμενο ημερολόγιο, να δει τα δεδομένα κυκλοφορίας για το διάστημα που έχει επιλέξει, να φιλτράρει τα δεδομένα αυτά και αν το επιθυμεί να τα κατεβάσει σε αρχείο. Η επιλογή του χρονικού διαστήματος γίνεται μέσα από ένα ημερολόγιο (Listing D.31). Για την δημιουργία του κάνουμε χρήση ενός στοιχείου που μας παρέχει η material library για την δημιουργία ημερολογίων όπως φαίνεται στο παρακάτω τμήμα κώδικα.

Listing D.31: Δημιουργία αναδυόμενου ημερολογίου

```

1     <mat-form-field>
2         <mat-label>Enter a date range</mat-label>
3         <mat-date-range-input [rangePicker]="picker">
4             <input matStartDate (dateChange)="onStartDateChange(↵
5                 $event)" placeholder="Start date">
6             <input matEndDate (dateChange)="onEndDateChange($event)"↵
7                 placeholder="End date">
8         </mat-date-range-input>
9         <mat-datepicker-toggle matIconSuffix [for]="picker"></mat-↵
10             datepicker-toggle>
11         <mat-date-range-picker touchUi="true" #picker></mat-date-↵
12             range-picker>
13     </mat-form-field>

```

Στο ημερολόγιο αυτό του έχουμε αναθέσει δύο input. Το startDate και το endDate κάθε φορά που ο χρήστης επιλέγει μια ημερομηνία καλείται και η αντίστοιχη μέθοδος. Σκοπός των μεθόδων αυτών είναι να μετατρέψουν την ημερομηνία που έχει δώσει ο χρήστης σε μια συγκεκριμένη μορφή ώστε έπειτα όπως θα μελετήσουμε και παρακάτω να μπορέσουμε να την περάσουμε σαν παράμετρο όταν θα καλέσουμε το service για να λάβουμε τα δεδομένα.

Στο σημείο αυτό να αναφέρουμε ότι κατά την ανάπτυξη του frontend χρησιμοποιήθηκαν αρκετές μεταβλητές τύπου Boolean με σκοπό να αναγνωρίζουμε την κατάσταση των δεδομένων, δηλαδή αν έχουμε λάβει τα δεδομένα, αν δεν υπάρχουν τα δεδομένα ή αν η εφαρμογή λαμβάνει τα δεδομένα. Οι μεταβλητές αυτές είναι οι `isDownloading`, `updateTableContent`, `noAvailableData`

Listing D.32: Μέθοδοι για το format του input του ημερολογίου

```

1   onStartDateChange(selectedDate: MatDatepickerInputEvent<any, any>) ↔
    {
2   if (selectedDate.value) {
3       this.noAvailableData = false;
4       this.formattedStartDate = selectedDate.value.toLocaleDateString()↔
        ();
5       console.log('Selected Start Date:', this.formattedStartDate);
6   }
7   }
8
9   onEndDateChange(selectedDate: MatDatepickerInputEvent<any, any>) {
10      if (selectedDate.value) {
11          this.formattedEndDate = selectedDate.value.toLocaleDateString()↔
              ;
12          console.log('Selected End Date:', this.formattedEndDate);
13          this.fetchData();
14      }
15  }
```

Αφού ο χρήστης επιλέξει και την τελική ημερομηνία θα γίνει κλήση της μεθόδου `fetchData` (Listing D.33). Στην μέθοδο αυτή θα περάσουμε σε HTTP παραμέτρους τις ημερομηνίες που λάβαμε από τον χρήστη και θα καλέσουμε από το `TableService` (Listing D.29) που περιγράψαμε και προηγουμένως την `fetchTableData`. Έπειτα το service θα κάνει κλήση στο backend, συγκεκριμένα στο endpoint `/gov`, ο Controller θα συλλέξει τα δεδομένα από τον `WebClient` (Listing D.5) και θα τα επιστρέψει στο frontend σε μια λίστα, όπου και στην συνέχεια θα γίνει η παρουσίαση τους σε πίνακα.

Listing D.33: *Κλήση του service για την συλλογή δεδομένων του πίνακα*

```

1  fetchData() {
2      this.updateTableContent = true;
3      this.noAvailableData = false;
4      let params = new HttpParams();
5      params = params.append('startDate', this.formattedStartDate);
6      params = params.append('endDate', this.formattedEndDate);
7      this.tableService.fetchTableData(params).subscribe(
8          (data: TableItem[]) => {
9              this.dataSource = new MatTableDataSource<TableItem>(data);
10             this.dataSource.paginator = this.paginator;
11             this.dataSource.sort = this.sort;
12             this.updateTableContent = false;
13
14             if(this.dataSource.data.length === 0){
15                 this.noAvailableData = true;
16             }
17
18             },
19             (error) => {
20                 console.error('Error fetching table data:', error);
21             });
22     }

```

Η αναπαράσταση του πίνακα γίνεται μέσα από την απάντηση που λάβαμε. Πιο συγκεκριμένα όπως μελετήσαμε και προηγουμένως όταν λαμβάνουμε μια απάντηση από το backend την μετατρέπουμε σε αντικείμενα του interface TableItem. Στο πίνακα προβάλλουμε σε κάθε στήλη το αντίστοιχο πεδίο κάθε αντικειμένου.

Listing D.34: *Κλήση του service για την συλλογή δεδομένων του πίνακα*

```

1  <div class="mat-elevation-z8">
2      <table mat-table [dataSource]="dataSource" matSort class="mat-↵
          elevation-z8">
3          <!-- Date and Time Column -->
4          <ng-container matColumnDef="appprocesstime">
5              <th mat-header-cell *matHeaderCellDef mat-sort-header>Date and ↵
                  Time</th>
6              <td mat-cell *matCellDef="let element">{{ element.↵

```

```

        appprocesstime }}</td>
7    </ng-container>
8
9    <!-- Road Name Column -->
10   <ng-container matColumnDef="road_name">
11       <th mat-header-cell *matHeaderCellDef mat-sort-header>Road Name<↵
        </th>
12       <td mat-cell *matCellDef="let element">{{ element.road_name }}<↵
        </td>
13   </ng-container>
14
15   <!-- Road Info Column -->
16   <ng-container matColumnDef="road_info">
17       <th mat-header-cell *matHeaderCellDef mat-sort-header>Road Info<↵
        </th>
18       <td mat-cell *matCellDef="let element">{{ element.road_info }}<↵
        </td>
19   </ng-container>
20
21   <!-- Counted Cars Column -->
22   <ng-container matColumnDef="countedcars">
23       <th mat-header-cell *matHeaderCellDef mat-sort-header>Cars</th>
24       <td mat-cell *matCellDef="let element">{{ element.countedcars <↵
        }}</td>
25   </ng-container>
26
27   <!-- Average Speed Column -->
28   <ng-container matColumnDef="average_speed">
29       <th mat-header-cell *matHeaderCellDef mat-sort-header>Average <↵
        Speed</th>
30       <td mat-cell *matCellDef="let element">{{ element.average_speed<↵
        }}</td>
31   </ng-container>
32
33   <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
34   <tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>
        >
35 </table>

```

```

36
37   <mat-paginator [pageSizeOptions]="[5, 10, 25, 50, 100]" [pageSize]=↵
      "5" showFirstLastButtons></mat-paginator>
38 </div>

```

Για να μπορέσουμε να φιλτράρουμε τα δεδομένα που έχουμε λάβει υλοποιήθηκε η μέθοδος `applyFilter` (Listing D.35). Η συγκεκριμένη μέθοδος λαμβάνει σε κείμενο το φίλτρο που έχει πληκτρολογήσει ο χρήστης και περιορίζει τα δεδομένα που προβάλλονται με βάση αυτό

Listing D.35: Μέθοδος για το φιλτράρισμα των δεδομένων του πίνακα

```

1   applyFilter(event: Event) {
2     const filterValue = (event.target as HTMLInputElement).value;
3     this.dataSource.filter = filterValue.trim().toLowerCase();
4   }

```

Η επιλογή αυτή εμφανίζεται στην οθόνη του χρήστη σαν text box μέσα σε ένα επεκτάσιμο πάνελ. Με το που ξεκινήσει να πληκτρολογεί ο χρήστης θα καλείτε η `applyFilter` (Listing D.35) και θα ανανεώνει το περιεχόμενο του πίνακα.

Listing D.36: Φίλτρο πίνακα

```

1 <mat-expansion-panel>
2   <mat-expansion-panel-header>Toggle to filter data</mat-expansion-↵
      panel-header>
3   <mat-form-field appearance="outline" style="width: 100%">
4     <mat-icon matSuffix>settings</mat-icon>
5     <mat-label>Filter table content</mat-label>
6     <input matInput (keyup)="applyFilter($event)" placeholder="Filter ↵
      based on street name, street info, average speed" #input>
7   </mat-form-field>
8 </mat-expansion-panel>

```

Η λήψη των δεδομένων γίνεται από την `downloadFile` (Listing D.37) η οποία παίρνει σαν παράμετρο ένα string που θα περιέχει το όνομα του αρχείου που θέλει ο χρήστης να κατεβάσει. Αφού γίνει κλήση της μέθοδος αυτή θα δώσει στο service τις ημερομηνίες που έχει επιλέξει ο χρήστης και τον τύπο του αρχείου που θέλει να κατεβάσει. Έπειτα το service θα επικοινωνήσει με το Backend, το backend θα επιστρέψει το αρχείο (blob) και ο περιηγητής θα το κατεβάσει.

Listing D.37: Μέθοδος για το κατέβασμα του αρχείου

```

1  async downloadFile(fileType: string){
2    this.noAvailableData = false;
3    this.isDownloading = true;
4    let params = new HttpParams();
5    params = params.append('startDate', this.formattedStartDate);
6    params = params.append('endDate', this.formattedEndDate);
7    this.tableService.downloadTableData(fileType, params).subscribe(↵
      blob => {
8      const url = window.URL.createObjectURL(blob);
9      const a = document.createElement('a');
10     a.href = url;
11     a.download = 'Data-Overview.'+fileType;
12     document.body.appendChild(a);
13     a.click();
14     document.body.removeChild(a);
15     window.URL.revokeObjectURL(url);
16     this.isDownloading = false;
17   });
18 }

```

Ο τρόπος που καλούμε την μέθοδο αυτή γίνεται μέσα από την HTML όπως φαίνεται στο παρακάτω τμήμα κώδικα. Ανάλογα με πιο κουμπί πατήσει ο χρήστης θα καλεστεί η μέθοδος downloadFile (Listing D.37) και θα λάβει και την αντίστοιχη παράμετρο

Listing D.38: Κλήση της downloadFile

```

1  <mat-menu #menu="matMenu">
2    <button mat-menu-item (click)="downloadFile('xml')">XML</↵
      button>
3    <button mat-menu-item (click)="downloadFile('json')">JSON</↵
      button>
4    <button mat-menu-item (click)="downloadFile('xlsx')">Excel</↵
      button>
5  </mat-menu>

```

D.2.1.4 Οπτικοποίηση Δεδομένων

Την αναπαράσταση των δεδομένων σε γραφήματα την αναλαμβάνει ο Dashboard component. Σαν τρόπο λειτουργίας φέρει αρκετά κοινά χαρακτηριστικά με τον component του Data Overview. Πιο αναλυτικά ο τρόπος που λαμβάνουμε τα δεδομένα είναι ίδιος. Ο χρήστης θα επιλέξει τις ημερομηνίες, οι ημερομηνίες αυτές θα προσαρμοστούν σε μια διαφορετική μορφή, θα τις περάσουμε στο service και το service αφού λάβει την απάντηση από το backend θα τις επιστρέψει στον component για να γίνει η γραφική αναπαράσταση τους.

Μέσα από την εφαρμογή προβάλουμε τα δεδομένα μας σε τρεις τύπους διαγραμμάτων. Χρησιμοποιούμε διαγράμματα πίτας, διαγράμματα γραμμών και ραβδόγραμματα. Κάθε ένα από αυτά έχει την δική του ξεχωριστή υλοποίηση όπου θα αναλύσουμε ξεχωριστά.

Το ραβδόγραμμα είναι αυτό που θα μας παρουσιάσει τους δρόμους με την μεγαλύτερη ή χαμηλότερη κίνηση. Στον άξονα x αναφέρει τα ονόματα των δρόμων, ενώ στον άξονα y τον συνολικό αριθμό καταγεγραμμένων οχημάτων που πέρασαν από εκείνο τον δρόμο. Όταν ο χρήστης σύρει το ποντίκι πάνω από μια μπάρα θα δει σχετικές πληροφορίες για την καταγραφή αυτή. Ως υλοποίηση δημιουργούμε ένα αντικείμενο της βιβλιοθήκης ChartJS. Στο τμήμα datasets καθορίζουμε τα δεδομένα που θέλουμε να προβάλουμε μαζί με κάποιες ρυθμίσεις για την εμφάνιση του διαγράμματος. Στο τμήμα options καθορίζουμε ότι το διάγραμμα μας θα είναι responsive, δηλαδή να μπορεί να ανταποκρίνεται στις ενεργείες του χρήστη όταν αλληλοεπιδρά με αυτό, όπως το να εμφανίζει επιπλέον πληροφορία όταν το ποντίκι βρίσκεται πάνω από κάποια μπάρα. Στο callbacks θέτουμε την πληροφορία που θα προβάλει στον χρήστη, όταν αυτός αλληλοεπιδρά με το διάγραμμα. Αυτό που καθορίζουμε στο συγκεκριμένο παράδειγμα είναι ότι η πληροφορία που θα εμφανιστεί θα περιέχει

1. Το όνομα του δρόμου
2. Τον ακριβές αριθμό οχημάτων που καταγράφηκαν
3. Το timestamp τότε έγινε η καταγραφή αυτή

Listing D.39: Δημιουργία Bar Chart

```
1   this.chart = new Chart( 'bar-chart ', {  
2       type: 'bar ',  
3       data: {
```

```
4      labels: labels,
5      datasets: [
6        {
7          label: 'Cars passed by',
8          data: countedCars,
9          borderWidth: 1,
10         borderColor: boarderColor,
11         backgroundColor: backgroundColor,
12       },
13     ],
14   },
15   options: {
16     responsive: true,
17     scales: {
18       y: {
19         beginAtZero: true,
20       },
21     },
22     plugins: {
23       tooltip: {
24         callbacks: {
25           label: (context) => {
26             const label = context.label || '';
27             const timestamp = this.chartTimestamp[context.dataIndex];
28             const formattedTimestamp = new Date(timestamp).toLocaleString();
29             return `${label}: ${context.parsed.y} (Timestamp: ${formattedTimestamp})`;
30           },
31         },
32       },
33     },
34   },
35 });
36 }
```

Η παρουσίαση του στο UI της εφαρμογής γίνεται μέσα από την HTML όπως φαίνεται στο παρακάτω τμήμα κώδικα. Στο πάνω τμήμα θέτουμε τον τίτλο του διαγράμματος καθώς και μια συνοπτική περιγραφή για το τι μας παρουσιάζει. Στην περίπτωση που η εφαρμογή λαμβάνει τα δεδομένα θα εμφανιστεί το σχετικό animation για να δείξει στον χρήστη ότι το αίτημα του επεξεργάζεται. Αφού ολοκληρωθεί η λήψη των δεδομένων μέσα από το `<canvas class="canvas" id="bar-chart"></canvas>` θα εμφανιστεί το διάγραμμα. Έπειτα θα εμφανιστεί και η σχετική ένδειξη από κάτω ότι τα δεδομένα είναι ανανεωμένα.

Listing D.40: Εμφάνιση του Bar Chart

```

1  <mat-card class="right-card">
2    <mat-card-content>
3      <mat-card-title>Vehicles on street</mat-card-title>
4      <mat-card-subtitle>Visualizing the number of vehicles on the ↵
        street</mat-card-subtitle>
5      <ng-container *ngIf="isLoadingData">
6        <div class="loading-container">
7          <div class="dot dot1"></div>
8          <div class="dot dot2"></div>
9          <div class="dot dot3"></div>
10         </div>
11       </ng-container>
12       <div class="dashboard-content">
13         <canvas class="canvas" id="bar-chart"></canvas>
14       </div>
15     </mat-card-content>
16     <ng-container *ngIf="showCardFooter">
17       <mat-divider></mat-divider>
18       <mat-card-footer><i class="material-icons" style="font-size↵
        :13px">update</i>Updated</mat-card-footer>
19     </ng-container>
20   </mat-card>

```

Αντίστοιχη λογική ακολουθούν και τα διάγραμμα γραμμής και πίτας. Η μόνη διαφορά τους είναι ότι στον τύπο του διαγράμματος τοποθετούμε τις αντίστοιχες τιμές line-chart και pie-chart.

Σημαντικό είναι να αναφέρουμε κάποια τμήματα κώδικα για το πως γίνεται η συλλογή των δεδομένων για την οπτικοποίηση. Όπως αναφέρθηκε και προηγουμένως τα δεδομένα τα λαμβ

βάνουμε σε αντικείμενα της διεπαφής `TableItem`. Πιο αναλυτικά τα αντικείμενα αυτά θα περιέχουν τα εξής πεδία.

- **roadname:** Το όνομα του δρόμου
- **roadinfo:** Πληροφορίες για τον δρόμο αυτό
- **appprocesstime:** Το Timestamp της καταγραφής
- **countedcars:** Αριθμός οχημάτων που πέρασαν από τον δρόμο αυτό
- **averagespeed:** Μέση ταχύτητα οχημάτων

Κάθε ένα από αυτά τα πεδία τα αποθηκεύουμε σε λίστες και τα δίνουμε σαν dataset στα διαγράμματα. Επίσης με κάθε επιλογή ημερομηνίας ο χρήστης θα λάβει και τα δεδομένα που αφορούν την μεγαλύτερη κίνηση στο οδικό δίκτυο και την χαμηλότερη ταυτόχρονα. Αυτό γίνεται για να συλλέξουμε κατευθείαν όλη την πληροφορία που θα προβληθεί και να δώσουμε την δυνατότητα στον χρήστη να αλληλοεπιδρά με τις διαφορετικές ενότητες του dashboard γρήγορα χωρίς να υπάρχουν καθυστερήσεις. Η εναλλαγή των dataset που παίρνουν τα διαγράμματα γίνεται με βάση την επιλογή που θα δώσει ο χρήστης στο φίλτρο πάνω από το Dashboard. Για παράδειγμα όταν επιλέξει να δει τα δεδομένα που αφορούν την μέτρια έως χαμηλή κίνηση στους δρόμους οι components που φτιάχνουν τα διαγράμματα θα λάβουν σαν παράμετρο την λίστα με τους δρόμους που έχουν χαμηλή κίνηση. Την λίστα αυτή την εισαγάγουμε στο πεδίο datasets του αντικειμένου ChartJS και έπειτα γίνεται η οπτικοποίηση των αποτελεσμάτων.

Listing D.41: Συλλογή δεδομένων για τα διαγράμματα υψηλής κίνησης

```

1 getDashBoardChartData(startDate: string, endDate: string): void {
2     this.isLoadingData = true;
3     this.showCardFooter = false;
4
5     let params = new HttpParams();
6     params = params.append('startDate', startDate);
7     params = params.append('endDate', endDate);
8
9     this.dashboardService.getChartData(params).subscribe(data => {
10        this.ChartLabels = data.map((data: { road_name: any; }) => data.road_name);

```

```

11     this.ChartCountedCars = data.map((data: { countedcars: any; }) ↵
        => data.countedcars);
12     this.ChartSpeed = data.map((data: { average_speed: any; }) => ↵
        data.average_speed);
13     this.ChartTimeStamp = data.map((data: { appprocesstime: any; })↵
        => data.appprocesstime);
14
15     // For Table Divider
16     this.createChartsForDashboard(this.ChartLabels, this.↵
        ChartCountedCars, this.ChartSpeed, '#e16d84', '#eca6b5');
17     this.onButtonClick('Highest Congestion');
18 });
19 this.isLoadingData = false;
20 this.showCardFooter = true;
21 }

```

Listing D.42: Συλλογή δεδομένων για τα διαγράμματα μέτριας έως και χαμηλής κίνησης

```

1 getDashBoardChartLowCongestionData(startDate: string, endDate: string↵
    ): void {
2     this.isLoadingData = true;
3     this.showCardFooter = false;
4
5     let params = new HttpParams();
6     params = params.append('startDate', startDate);
7     params = params.append('endDate', endDate);
8
9     this.dashboardService.getLowCongestionData(params).subscribe(data↵
        => {
10         this.LowChartLabels = data.map((data: { road_name: any; }) => ↵
            data.road_name);
11         this.LowChartCountedCars = data.map((data: { countedcars: any; ↵
            }) => data.countedcars);
12         this.LowChartSpeed = data.map((data: { average_speed: any; }) ↵
            => data.average_speed);
13         this.LowChartTimeStamp = data.map((data: { appprocesstime: any;↵
            }) => data.appprocesstime);
14

```

```
15     // For Table Divider
16     this.isLoadingData = false;
17     this.showCardFooter = true;
18   });
19 }
```

D.3 Containerization

Το Containerization πραγματοποιείται μέσα από το Docker που χρησιμοποιείται για να συσκευάσει τις εφαρμογές και όλες τις εξαρτήσεις της μαζί σε μορφή containers, εξασφαλίζοντας ότι η εφαρμογή λειτουργεί σωστά σε οποιοδήποτε περιβάλλον. Τα containers είναι απομονωμένα μεταξύ τους και τρέχουν σαν ανεξάρτητες διεργασίες. Με αυτό τον τρόπο επιτρέπει στους προγραμματιστές να δημιουργούν, να αναπτύσσουν και να εκτελούν εφαρμογές με μεγαλύτερη αποτελεσματικότητα, ασφάλεια και ευελιξία. Παράλληλα το Docker είναι συμβατό με πολλές πλατφόρμες cloud, όπως το Azure και το AWS, πράγμα που καθιστά εύκολη την μεταφορά και το deployment των εφαρμογών σε αυτές τις πλατφόρμες.

D.3.1 Docker και Docker Compose

Το Dockerfile και το Docker Compose αποτελούν τα εργαλεία με τα οποία επιτυγχάνουμε το Containerization. Σαν εργαλεία εργάζονται από κοινού για να δημιουργήσουν και να τρέξουν την εφαρμογή σε containers, επιτρέποντας την απομόνωση, την επαναληψιμότητα και την κλιμακωσιμότητα. Μέσα από τα επόμενα υποκεφάλαια θα μελετήσουμε τον τρόπο λειτουργίας και χρήσης τους στην εφαρμογή

D.3.1.1 Dockerfile

Το Dockerfile είναι ένα αρχείο κειμένου που περιέχει οδηγίες για την δημιουργία ενός Docker image. Κάθε εφαρμογή θα έχει το δικό της Dockerfile, το οποίο θα περιγράφει πώς να δημιουργηθεί ένα image που μπορεί να τρέξει την εφαρμογή. Στην εφαρμογή που μελετάμε έχουν δημιουργηθεί δύο Dockerfile, ένα για το Backend και ένα για το Frontend

Με βασικό image το `openjdk:17-jdk-alpine` το Dockerfile που έχει δημιουργηθεί για το Backend (Listing D.43) θα αντιγράψει το Jar αρχείο που υπάρχει στον target φάκελο και θα το εκτελέσει όταν εκτελεστεί και ο container

Listing D.43: *Spring Boot Backend Dockerfile*

```

1 FROM openjdk:17-jdk-alpine
2
3 WORKDIR /app
4 COPY target/backend-0.0.1-SNAPSHOT.jar app.jar
5 ENTRYPOINT ["java", "-jar", "app.jar"]

```

Στο Dockerfile της Angular ορίζουμε σαν βασικό image το `node:20.9.0`. Αυτό το Dockerfile (Listing D.44) ορίζει ότι ο container θα αντιγράψει το περιεχόμενο του `package.json` και `package-lock.json`, θα κάνει εγκατάσταση την εφαρμογής με το `npm install` και θα ενημερώσει τον container ότι η εφαρμογή αυτή ακούει στην θύρα 4200

Listing D.44: *Angular Frontend Dockerfile*

```

1 FROM node:20.9.0
2
3 WORKDIR /app
4
5 COPY package*.json ./
6
7 RUN npm install -g @angular/cli
8 RUN npm install
9 COPY . .
10 EXPOSE 4200
11 CMD ["ng", "serve", "--host", "0.0.0.0"]

```

D.3.1.2 Docker Compose

Το Docker Compose είναι ένα εργαλείο που χρησιμοποιείται για τον ορισμό και την εκτέλεση πολλαπλών containers. Η ρύθμιση του γίνεται από ένα αρχείο `docker-compose.yml` (Listing D.45), το οποίο θα περιγράφει πώς να εκτελεστούν τα δύο containers μαζί, πώς να διαμοιραστούν τα δίκτυα και τα volumes, και πώς να ρυθμιστούν τα enviroments.

Στο Docker Compose της εφαρμογής ορίζουμε δύο containers τον backend και τον frontend οι οποίοι βρίσκονται κάτω από το ίδιο bridge δίκτυο, ώστε να μπορούν να επικοινωνούν μεταξύ τους. Και τα δύο containers χτίζονται από το εκάστοτε Dockerfile καθώς και για κάθε εφαρμογή ορίζουμε και την θύρα στην οποία θα ακούει.

Listing D.45: *Docker Compose για την διαχείριση πολλαπλών container*

```
1 version: '3.8'
2
3 services:
4
5   backend:
6     container_name: backend
7     build:
8       context: ./backend
9     ports:
10      - "8080:8082"
11     networks:
12      - backend-network
13
14   frontend:
15     container_name: frontend
16     build:
17       context: ./frontend
18     ports:
19      - "4200:4200"
20     depends_on:
21      - backend
22     networks:
23      - backend-network
24
25 networks:
26   backend-network:
27     driver: bridge
```

Κεφάλαιο Ε

Συμπεράσματα και Μελλοντικές κατευθύνσεις

Η εργασία πραγματεύεται τον σχεδιασμό και την ανάπτυξη μια διαδραστικής εφαρμογής για την οπτικοποίηση της κατάστασης του οδικού δικτύου σε έξυπνες πόλεις. Αυτή η εφαρμογή αξιοποιεί δημόσια δεδομένα από ένα API που μας παρέχεται από το γον για την πόλη της Αθήνας. Τα δεδομένα αυτά συλλέγονται και δημοσιοποιούνται από αρμόδιους φορείς όπως είναι η Περιφέρεια Αττικής. Μέσα από την εφαρμογή δίνουμε έμφαση στην διαδραστικότητα μέσα από διάφορα αντικείμενα που παρέχουν ανάδραση προς τον χρήστη. Όσο αφορά τον σχεδιασμό, στόχος ήταν η ανάπτυξη μιας επεκτάσιμης εφαρμογής η οποία θα μπορεί να ενσωματωθεί και να προσαρμοστεί με βάση της ανάγκες κάθε πόλης.

Ε.1 Συμπεράσματα

Στο πλαίσιο της υλοποίησης της εφαρμογής χρησιμοποιήσαμε ανοιχτά δεδομένα που παρέχονται από την Κυβέρνηση, δείχνοντας πως μπορούν να χρησιμοποιηθούν δημόσια δεδομένα για να συμβάλουν στην δημιουργία ενός βιώσιμου σχεδιασμού της αστικής κοινότητας. Επιπλέον με την χρήση των δεδομένων αυτών βοηθάμε τους χρήστες να αντλήσουν πληροφορία και να εντοπίσουν πιθανά μοτίβα στο οδικό δίκτυο της Αθήνας.

Δίνοντας έμφαση στην οπτικοποίηση της κατάστασης στο οδικό δίκτυο, χρησιμοποιώντας δι-

αγράμματα και άλλα διαδραστικά εργαλεία, στοχεύουμε να παρέχουμε γνώση στους αρμόδιους φορείς η οποία θα συμβάλει στην βελτίωση του οδικού δικτύου. Με την χρήση των εργαλείων αυτών επιτρέπουμε στον χρήστη να εντοπίσει μοτίβα στο οδικό δίκτυο, να δει πληροφορίες για τον κάθε δρόμο και την κατάταξη του μέσου όρου ταχύτητας. Παράλληλα με την διάθεση των δεδομένων αυτών σε αρχεία κάνουμε λόγο για διαθέσιμα και προσβάσιμα δεδομένα προς τον χρήστη, που μπορεί να τα χρησιμοποιήσει για περεταίρω μελέτη. Παράλληλα παρέχουμε στον οποιοδήποτε σχεδιαστή συστήματος την δυνατότητα να χρησιμοποιήσει τμήμα της λειτουργικότητας της εφαρμογής μας μέσα από τα Rest API του backend, τονίζοντας με αυτό τον τρόπο την επεκτασιμότητα της εφαρμογής.

Η υλοποίηση της εφαρμογής αποτέλεσε μια μαθησιακή εμπειρία μέσα από την οποία έγινε χρήση και μελέτη εργαλείων όπως Java, Spring Boot, Maven, Angular, Angular Material UI, ChartJS και Docker. Η διαδικασία αυτή αποκάλυψε την κυκλική φύση της τεχνολογικής προόδου, καθιστώντας εμφανές ότι ο προγραμματισμός είναι ένας δυναμικός τομέας που απαιτεί συνεχή μάθηση και προσαρμογή καθώς οι τεχνολογίες συνεχώς βελτιώνονται και αυξάνονται.

Επιθυμητό θα ήταν η χρήση της εφαρμογής να εναρμονίζεται με το όραμα της έξυπνης πόλης και να αποτελεί σημαντικό εργαλείο των κοινοτήτων και αναλυτών δεδομένων για την καταπολέμηση της κυκλοφοριακής συμφόρησης το οποίο θα συμβάλει στην βελτίωση του τρόπου ζωής των πολιτών.

Ε.2 Μελλοντικές κατευθύνσεις

Η εφαρμογή που παρουσιάζεται σε αυτή την εργασία αποδεικνύει την ανάγκη για την αξιοποίηση δεδομένων κυκλοφορίας στο οδικό δίκτυο με στόχο την βελτίωση του βιοτικού επιπέδου στα αστικά κέντρα. Αν και το εγχείρημα αυτό βρίσκεται σε πρώιμο στάδιο η εφαρμογή αυτή αναδεικνύει τα θεμέλια πάνω στα οποία μπορούν να ενσωματωθούν περισσότερα χαρακτηριστικά και δεδομένα για περισσότερες πόλεις.

Η εφαρμογή αξιοποιεί δεδομένα προς το παρόν από την πόλη της Αθήνας. Σημαντικό θα ήταν να εξερευνηθεί και η δυνατότητα ενσωμάτωσης δεδομένων και για διαφορετικές πόλεις και η λήψη τους από διαφορετικές πηγές όπως είναι το Google Maps, Bing Maps και TomTom.

Εξίσου σημαντική θα ήταν και η αξιοποίηση δεδομένων σε πραγματικό χρόνο. Οι υπηρεσίες που αναφέρθηκαν προηγουμένων έχουν την δυνατότητα να μας παρέχουν γνώση για την κυκλοφοριακή κατάσταση καλύπτοντας το μεγαλύτερο τμήμα του οδικού δικτύου σε ολόκληρο τον κόσμο. Φυσικά τα δεδομένα μας μπορούν να παρέχονται και από διάφορους αισθητήρες. Η ιδέα των έξυπνων πόλεων αντικατοπτρίζεται από ένα μεγάλο δίκτυο αισθητήρων διατρεγμένο μέσα σε μια πόλη. Πόλεις και χώρες όπως η Νέα Υόρκη, η Σιγκαπούρη και η Ιαπωνία έχουν επενδύσει σε τέτοια δίκτυα παρέχοντας γνώση για την κατάσταση στο οδικό δίκτυο σε πραγματικό χρόνο. Ασφαλώς μια τέτοια προσέγγιση για την λήψη των δεδομένων απαιτεί τόσο την απαραίτητη υποδομή μέσα σε ένα αστικό κέντρο όσο και την διάθεση των δεδομένων που συλλέγονται προς στο κοινό.

Όσο αφορά την λειτουργικότητα της εφαρμογής χρήσιμο θα ήταν να προστεθεί επιπλέον πληροφορία για τα οδικά δίκτυα. Η πληροφορία αυτή μπορεί να αναφέρεται στον τύπο του οχήματος για το οποίο έχει γίνει η καταγραφή, για παράδειγμα αν είναι μοτοσυκλέτα, λεωφορείο, ποδήλατο κ.τ.λ.. Μια άλλη σημαντική παράμετρος θα ήταν και αυτή των τροχαίων ατυχημάτων η οποία θα παρείχε γνώση για την ασφάλεια του οδικού δικτύου.

Σκοπός της εργασίας είναι να τονίσει την ανάγκη για περαιτέρω έρευνα και ανάπτυξη σχετικά με την οπτικοποίηση της κατάστασης στο οδικό δίκτυο σε έξυπνες πόλεις, παράγοντας που σίγουρα θα συμβάλει στη βελτίωση της ποιότητας ζωής των πολιτών και στη δημιουργία πιο βιώσιμων και ανθεκτικών αστικών περιβαλλόντων.

Βιβλιογραφία

- [1] Alexandra Jonker Alice Gomstyn. *What is a smart city?* Tech. rep. IBM, 2023.
- [2] Attiki Odos Traffic Management and Motorway Maintenance. 2024. URL: <https://www.aodos.gr/en/motorway-operation/traffic-management-and-motorway-maintenance/>.
- [3] Mohammad Ahmadi Babak Bashari Rad Harrison John Bhatti. “An Introduction to Docker and Analysis of its Performance”. In: (2017).
- [4] Mengting Bai et al. “PrePCT: Traffic congestion prediction in smart cities with relative position congestion tensor”. In: *Neurocomputing* 444 (2021), pp. 147–157. ISSN: 0925-2312.
- [5] Charles R. Berger and Egan Smith. “Intelligent Transportation Systems Provide Operational Benefits for New York Metropolitan Area Roadways: A Systems Engineering Approach”. In: *2007 IEEE Long Island Systems, Applications and Technology Conference*. 2007, pp. 1–8. DOI: 10.1109/LISAT.2007.4312628.
- [6] Laura Po Chiara Bachechi. “Traffic analysis in a smart city”. In: *International Conference on Web Intelligence* (2019).
- [7] Nuri Alperen Kose Damilola Oladimeji Khushi Gupta et al. “Smart Transportation: An Overview of Technologies and Applications”. In: *Sensors* (2023).
- [8] Gaurav Pushkarn Ekaterina Dmitrieva Ashish Pathan. “Real-Time Traffic Management in Smart Cities: Insights from the Traffic Management Simulation and Impact Analysis”. In: *BIO Web of Conferences* (2024).
- [9] Eurostat Statistics Explained. 2023. URL: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Transport_statistics_at_regional_level.

- [10] S A Evtiukov. “Smart Transport in road transport infrastructure”. In: *IOP Conference Series* (2020).
- [11] Mark Heckler. *Spring Boot Up Running, Building cloud native Java and Kotlin Applications*. O’Reilly Media, Inc, 2020.
- [12] Saskia Hollborn. *Intelligent Transport Systems in Japan*. 2002.
- [13] IBM. *What is Java Spring Boot?* Accessed: 2023-01-12. 2024. URL: <https://www.ibm.com/topics/java-spring-boot>.
- [14] David Palma Kaja Fjortoft Ystgaard Luigi Atzori et al. “Review of the theory, principles, and design requirements of human-centric Internet of Things (IoT)”. In: *Ambient Intelligence and Humanized Computing* (2023).
- [15] LTA Singapore Intelligent Transport System. 2024. URL: https://www.lta.gov.sg/content/ltagov/en/getting_around/driving_in_singapore/intelligent_transport_systems.html.
- [16] Apache Maven. *Maven – Maven Releases History*. Accessed: 2024-01-14. 2024. URL: <https://maven.apache.org/docs/history.html>.
- [17] Road safety in the EU: fatalities below pre-pandemic levels but progress remains too slow. 2023. URL: https://transport.ec.europa.eu/news-events/news/road-safety-eu-fatalities-below-pre-pandemic-levels-progress-remains-too-slow-2023-02-21_en.
- [18] Radouan Ait Mouha. *Journal of Data Analysis and Information Processing*. Scientific Research Publishing, 2024.
- [19] Keith O’Brien. *What is smart transportation*. Tech. rep. IBM, 2023.
- [20] 8 Smart Cities Lead the Way in Advanced Intelligent Transportation Systems. 2021. URL: <https://otonomo.io/blog/smart-cities-intelligent-transportation-systems/>.
- [21] Josh Long Phillip Webb Dave Syer, Stéphane Nicoll, and Marcel Overdijk Rob Winch Andy Wilkinson. “Spring Boot Reference Documentation”. In: (2024), p. 972.
- [22] Mohammed J. F. Alenazi Razan M. AlZoman. “A Comparative Study of Traffic Classification Techniques for Smart City Networks”. In: (2021).
- [23] Pivotal Software. *Spring Boot*. Accessed: 2023-01-12. 2024. URL: <https://spring.io/projects/spring-boot>.

- [24] TWI. *What is a Smart City - Defination and Examples*. Accessed: 2024-02-03. 2023. URL: <https://www.twi-global.com/technical-knowledge/faqs/what-is-a-smart-city>.
- [25] Balaji Varanasi. *Introducing Maven: A Build Tool for Todays Java Developers*. Apress Media LLC, 2019.
- [26] Jeremy Wilken. *Angular in Action*. Manning Publications Co, 2018.
- [27] Haipeng Yao et al. “Capsule Network Assisted IoT Traffic Classification Mechanism for Smart Cities”. In: *IEEE Internet of Things Journal* 6.5 (2019), pp. 7515–7525. DOI: 10.1109/JIOT.2019.2901348.
- [28] Wenlong Shang Zhihan Lv a. “Impacts of intelligent transportation systems on energy conservation and emission reduction of transport systems: A comprehensive review”. In: *KeAi* (2023).
- [29] ΓΙΑΝΝΑΚΟΥ ΑΘΗΝΑ ΑΝΔΡΙΚΟΠΟΥΛΟΥ ΕΛΕΝΗ. *ΠΟΛΗ ΚΑΙ ΠΟΛΕΟΔΟΜΙΚΕΣ ΠΡΑΚΤΙΚΕΣ ΓΙΑ ΤΗ ΒΙΩΣΙΜΗ ΑΣΤΙΚΗ ΑΝΑΠΤΥΞΗ*. ΚΡΙΤΙΚΗ, 2014.

Συντμήσεις

API	Application Programming Interface
AWS	Amazon Web Services
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
IoC	Inversion Of Control
IoT	Internet of Things
ICT	Information and Communication Technology
ITS	Intelligent Transport Systems
I/O	Input/Output

JAR	Java ARchive
JSON	JavaScript Object Notation
POM	Project Object Model
UI	User Interface
UTF-8	Unicode Transformation Format - 8 bit
XML	eXtensible Markup Language

Γλωσσάρι Ξενικών Όρων

Autoconfiguration	Αυτόματη διαμόρφωση
Animation	Κινούμενα σχέδια
Client-side	Πλευρά πελάτη
Dependencies	Εξαρτήσεις
Deploy	Ανάπτυξη
Dashboard	Πίνακας ελέγχου
Configuration	Διαμόρφωση
Component	Συστατικό
Services	Υπηρεσίες
Smart City	Έξυπνη Πόλη
Intergradable	Ενσωματώσιμο

Intelligent Transport System	Έξυπνο Σύστημα Μεταφορών
Internet Of Things	Διαδίκτυο των Πραγμάτων
Backend	Παρασκήνιο Εφαρμογής
Frontend	Προσκήνιο Εφαρμογής
Business Layer	Επίπεδο Επιχείρησης/Λογικής
Presentation Layer	Επίπεδο παρουσίασης
Build	Κατασκευή
Reactive	Ανταποκριτικό