

ΙΟΝΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ



Πτυχιακή Εργασία

Εξόρυξη Γνώσης Ροών Πληροφορίας Τιμής Εισιτηρίων Μέσων Μαζικής Μεταφοράς

Γεώργιος Μαργαρίτης

Αρ. Μητρώου: Π2015139

Επιβλέπων: Ιωάννης Καρύδης

Κέρκυρα, Σεπτέμβριος 2021

Επιβλέπων

Ιωάννης Καρύδης, *Επίκουρος Καθηγητής*
Ιόνιο Πανεπιστήμιο

Τριμελής Επιτροπή

Ιωάννης Καρύδης, *Επίκουρος Καθηγητής*
Ιόνιο Πανεπιστήμιο

Αγγελική Τσώχου, *Επίκουρη Καθηγήτρια*
Ιόνιο Πανεπιστήμιο

Κάτια - Λήδα Κερμανίδου, *Αναπληρώτρια Καθηγήτρια*
Ιόνιο Πανεπιστήμιο

Περίληψη

Η ραγδαία ανάπτυξη της τεχνολογίας σε συνδυασμό με την ωρίμανση του διαδικτύου, έχουν αποτελέσει κύριοι παράγοντες κατά την εκσυγχρόνιση της ανθρώπινης κοινωνίας. Σε ένα τέτοιο πλαίσιο ο όγκος της πληροφορίας που ανταλλάσσεται και αποθηκεύεται καθημερινά από τα ηλεκτρονικά συστήματα αυξάνεται συνεχώς. Για το λόγο αυτό υπάρχει ένα αυξανόμενο ενδιαφέρον για την ανακάλυψη γνώσης από δεδομένα μεγάλου όγκου, με στόχο την αξιοποίηση της για την αποτελεσματικότερη λήψη αποφάσεων. Η παρούσα πτυχιακή εργασία, εστιάζει στη διαδικασία εξόρυξης χρήσιμης πληροφορίας, από δεδομένα που αναφέρονται στον τομέα των Μέσων Μαζικής Μεταφοράς και συγκεκριμένα σε αυτόν, των αερομεταφορών.

Η εργασία αυτή δίνει μεγαλύτερη έμφαση στη σχεδίαση και την υλοποίηση ενός συστήματος που έχει τη δυνατότητα να συλλέξει και να αποθηκεύσει δεδομένα, να τα αναλύσει και να εξάγει γνώση από αυτά, ενώ στη συνέχεια να παρέχει τα ευρήματα στο ευρύ κοινό μέσω ενός Web API. Για την υλοποίηση του συστήματος αυτού, χρησιμοποιήθηκε η γλώσσα προγραμματισμού JavaScript, το περιβάλλον εκτέλεσης Node.js καθώς και το framework Express.js. Με σκοπό την αποθήκευση των δεδομένων, αξιοποιήθηκε το σχεσιακό σύστημα διαχείρισης βάσεων δεδομένων MySQL ενώ η εκτέλεση της εφαρμογής σε διαδικτυακό περιβάλλον πραγματοποιήθηκε μέσω της πλατφόρμας Heroku. Η δομή του συστήματος αποτελείται από τρεις ξεχωριστές υπηρεσίες, οι οποίες είναι υπεύθυνες για τη συλλογή και την ανάλυση δεδομένων καθώς για τη διάχυση των συμπερασμάτων.

Στα πλαίσια της παρούσας πτυχιακής εργασίας, πραγματοποιήθηκαν επίσης τέσσερα ερευνητικά πειράματα, με σκοπό την ανάδειξη της λειτουργίας του παραπάνω συστήματος. Πιο συγκεκριμένα, ο στόχος αυτού του σκέλους ήταν η εξόρυξη γνώσης σχετικά με την τιμή αεροπορικών εισιτηρίων. Η συλλογή των δεδομένων διήρκεσε 22 ημέρες και συλλέχθηκαν 9,926 εγγραφές. Ως πηγή άντλησης δεδομένων, επιλέχθηκε η αεροπορική εταιρεία Aegean Airlines. Μέσω αυτών των πειραμάτων εξάχθηκαν συμπεράσματα για συγκεκριμένα δρομολόγια που έχουν ως προορισμούς ελληνικές πόλεις, σχετικά με τη συμπεριφορά της μέσης τιμής των αεροπορικών εισιτηρίων.

Λέξεις Κλειδιά: Διαδικτυακές Εφαρμογές, Εξόρυξη Γνώσης από Δεδομένα, Ανάλυση Δεδομένων, Μέσα Μαζικής Μεταφοράς, Αεροπορικά Εισιτήρια, JavaScript, Node.js, Express.js, MySQL, Heroku, REST

Abstract

The rapid development of technology in combination with the never ending growth of the internet, have been major factors in the modernization of human society. In such a context, the volume of information exchanged and stored daily by electronic systems is constantly increasing. For this reason, there is a growing interest in the discovery of knowledge from large volumes of data, with the aim of utilizing it for more effective decision making. The present thesis focuses on the process of extracting useful information from data related to the field of public transport and specifically to that of aviation.

This work places more emphasis on designing and implementing a system that has the ability to collect and store data, analyze it and extract knowledge from it, while then providing the findings to the general public through a Web API. The JavaScript programming language, the Node.js runtime environment, and the Express.js framework were used to implement this system. In order to store the data, the MySQL relational database management system was utilized while the implementation of the application in an online environment was carried out through the Heroku platform. The structure of the system consists of three separate services, which are responsible for data collection and analysis, as well as for disseminating conclusions.

In the context of this thesis, four research experiments were also carried out, in order to highlight the operation of the system above. More specifically, the aim of this research was to extract knowledge about the behavior of airline ticket prices. The data collection lasted 22 days and 9,926 records were collected. The data was collected from a Greek airline company, Aegean Airlines. Through these experiments, conclusions were drawn for specific routes that have Greek cities as destinations, regarding the behavior of the average price of airline tickets.

Keywords: Web Applications, Data Mining, Knowledge Discovery in Databases, Data Analysis, Public Transport, Airfare, JavaScript, Node.js, Express.js, MySQL, Heroku, REST

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου κ. Ιωάννη Καρύδη, Επίκουρο Καθηγητή του Τμήματος Πληροφορικής, για την εμπιστοσύνη αλλά και την επιστημονική καθοδήγηση που μου παρείχε καθ' όλη την διάρκεια της διεξαγωγής της πτυχιακής μου εργασίας. Επίσης, θα ήθελα να ευχαριστήσω τους γονείς μου, τον αδερφό μου και την κοπέλα μου για την αμέριστη υποστήριξη που μου προσέφεραν όλα αυτά τα χρόνια.

Περιεχόμενα

Περίληψη	iii
Abstract	iv
Ευχαριστίες	v
Περιεχόμενα	vi
Κατάλογος Σχημάτων	viii
Κατάλογος Πινάκων	x
Κατάλογος Εξισώσεων	xi
Αποσπάσματα Κώδικα	xii
1 Εισαγωγή	1
1.1 Πρόβλημα - Σημαντικότητα του θέματος	1
1.2 Στόχος - Σκοπός	1
1.3 Διάρθρωση της Εργασίας	2
2 Τεχνολογικό Υπόβαθρο	3
2.1 Εξόρυξη Γνώσης από Δεδομένα	3
2.2 Συλλογή Δεδομένων	5
2.3 Βάση Δεδομένων	6
2.3.1 Επιλογή Τύπου Βάσης Δεδομένων	6
2.3.1 MySQL	8
2.4 JavaScript	9
2.5 Node.js	12
2.5.1 NPM	13
2.5.2 Event Loop	13
2.6 Express.js	15
2.7 Heroku	16
2.7.1 Heroku Dynos	17
2.7.2 ClearDB	18

2.8 HTTP	19
2.8.1 Μέθοδοι HTTP	20
2.8.2 Επικεφαλίδες HTTP	21
2.8.3 RESTful APIs	22
3 Σχεδιασμός	25
3.1 Καταγραφή απαιτήσεων	25
3.2 Σχεδίαση Βάσης Δεδομένων	26
3.2.1 Πίνακας Airports	28
3.2.2 Πίνακας Flights	29
3.3 Αρχιτεκτονική Συστήματος	29
3.3.1 Αρχιτεκτονική RESTful API	33
4 Υλοποίηση	36
4.1 Εγκατάσταση εργαλείων	36
4.1.1 Βιβλιοθήκες	37
4.1.2 Heroku	39
4.2 Υλοποίηση Βάσης Δεδομένων	40
4.3 Scheduling	42
4.4 Διαχείριση Σφαλμάτων	45
4.5 Συλλογή και Αποθήκευση Δεδομένων	46
4.6 Υλοποίηση RESTful API	55
5 Πειραματική Αποτίμηση	58
5.1 Ανάλυση Δεδομένων	58
5.2 Πείραμα 1	62
5.3 Πείραμα 2	64
5.4 Πείραμα 3	67
5.5 Πείραμα 4	70
6 Επίλογος	72
6.1 Σύνοψη και Συμπεράσματα	72
6.2 Μελλοντικές Επεκτάσεις	73
Βιβλιογραφία	74

Κατάλογος Σχημάτων

2.1	Στάδια του Event Loop	14
2.2	Παράδειγμα dyno formation	18
2.3	Δομικά στοιχεία του Uniform Resource Locator (URL)	19
3.1	Διάγραμμα περιπτώσεων χρήσης	26
3.2	Διάγραμμα Οντοτήτων-Συσχετίσεων	27
3.3	Σχήμα Crow's Foot	28
3.4	Διάγραμμα Ροής Δεδομένων Μηδενικού Επιπέδου	30
3.5	Διάγραμμα Ροής Δεδομένων Πρώτου Επιπέδου - Υπηρεσία Συλλογής Δεδομένων	31
3.6	Διάγραμμα Ροής Δεδομένων Πρώτου Επιπέδου - Υπηρεσία Ανάλυσης Δεδομένων	31
3.7	Διάγραμμα Ροής Δεδομένων Πρώτου Επιπέδου - Υπηρεσία Διάχυσης	32
3.8	Διάγραμμα Ροής Δεδομένων Πρώτου Επιπέδου	32
3.9	Αρχιτεκτονική των API endpoints	33
4.1	Παράδειγμα καταχώρησης configuration variables στο Heroku	39
4.2	Ορισμός μεσοδιαστήματος διεργασίας	45
5.1	Πείραμα 1 - Μεγαλύτερη διακύμανση τιμής σε ακριβότερα δρομολόγια	64
5.2	Πείραμα 1 - Αύξηση τιμής ανά εβδομάδα	64
5.3	Πείραμα 1 - Διακύμανση μηνιαίας μέσης τιμής	65
5.4	Πείραμα 2 - Εξέλιξη μέσης τιμής για τους προορισμούς Σαντορίνη, Κέρκυρα, με αναχώρηση από τη Ρώμη	66
5.5	Πείραμα 2 - Εξέλιξη μέσης τιμής για τους προορισμούς Σαντορίνη, Κέρκυρα, με αναχώρηση από το Λονδίνο	67
5.6	Πείραμα 2 - Εξέλιξη μέσης τιμής για τους προορισμούς Μήλο, Κέρκυρα, με αναχώρηση από το Βερολίνο	67
5.7	Πείραμα 2 - Εξέλιξη μέσης τιμής, Βερολίνο προς Κέρκυρα	68

5.8	Πείραμα 3 - Διαφοροποιήσεις μέσης τιμής, Αθήνα προς Θεσσαλονίκη (RT - OW)	70
5.9	Πείραμα 3 - Διαφοροποιήσεις μέσης τιμής, Θεσσαλονίκη προς Αθήνα (RT - OW)	70
5.10	Πείραμα 4 - Εξέλιξη μέσης τιμής ανά ημέρα συλλογής	71

Κατάλογος Πινάκων

3.1 Πίνακας Airports	28
3.2 Πίνακας Flights	29
3.3 Περιγραφή πεδίων απόκρισης	35
5.1 Πείραμα 1 - Στατιστικά στοιχεία δρομολογίων	65
5.2 Πείραμα 2 - Στατιστικά στοιχεία δρομολογίων	66
5.3 Πείραμα 3 - Στατιστικά στοιχεία δρομολογίων	69
5.4 Πείραμα 4 - Στατιστικά στοιχεία δρομολογίων	72

Κατάλογος Εξισώσεων

5.1 Μέση Τιμή	59
5.2.1 Διάμεσος – Περισσότερο πλήθος παρατηρήσεων	60
5.2.2 Διάμεσος – Ζυγό πλήθος παρατηρήσεων	60
5.3 Διακύμανση	60
5.4 Εύρος	60
5.5 Τυπική Απόκλιση	61

Αποσπάσματα Κώδικα

2.1 Αρχική δομή δεδομένων Aegean	6
2.2 Παράδειγμα promie hell	11
2.3 Παράδειγμα χρήσης του async/await	12
2.4 Παράδειγμα εγκατάστασης npm πακέτου	13
2.5 Παράδειγμα χρήσης Express.js	15
2.6 Παράδειγμα GET αίτησης	20
3.1 Υποθετικό παράδειγμα απόκρισης για το endpoint /results	36
4.1 Εντολές εγκατάστασης Node.js και npm	37
4.2 Εντολή εγκατάστασης Express.js	37
4.3 Εντολή εγκατάστασης του εργαλείου dotenv	38
4.4 Αρχείο .env	38
4.5 Ενσωμάτωση της βιβλιοθήκης dotenv	38
4.6 Ορισμός ευαίσθητων στοιχείων της βάσης δεδομένων	39
4.7 Εντολή εγκατάστασης της βιβλιοθήκης node-fetch	39
4.8 Παράδειγμα χρήσης της βιβλιοθήκης node-fetch	40
4.9 Εντολή εγκατάστασης Heroku	40
4.10 Εντολή σύνδεσης με το Heroku	40
4.11 Εντολή δημιουργίας απομακρυσμένου αποθετηρίου στο Heroku	40
4.12 Αρχείο procfile	41
4.13 Δημιουργία Βάσης Δεδομένων	42
4.14 Ενσωμάτωση της βιβλιοθήκης mysql	42
4.15 Αρχικοποίηση του Πίνακα Airports	43

4.16	Εντολή εγκατάστασης node-cron	44
4.17	Συνάρτηση scheduler	44
4.18	Ενσωμάτωση βιβλιοθήκης winston	47
4.19	Παράδειγμα χρήσης βιβλιοθήκης winston	47
4.20	Αρχικοποίηση airportGroups	48
4.21	Συνάρτηση processAirportGroups	49
4.22	Συνάρτηση handleFlightData	50
4.23	Συνάρτηση getAirports	51
4.24	Συνάρτηση getFlightData	54
4.25	Συνάρτηση getFlights	55
4.26	Συνάρτηση addFlights	55
4.27	Συνάρτηση storeData	56
4.28	Ενσωμάτωση Express.js	57
4.29	Δημιουργία API endpoints	58
5.1	Συναρτήσεις ανάλυσης δεδομένων	63

1 Εισαγωγή

1.1 Πρόβλημα - Σημαντικότητα του θέματος

Τα τελευταία χρόνια, λόγω της ταχύτατης ανάπτυξής της, η τεχνολογία έχει διεισδύσει σε όλες τις εκφάνσεις της ζωής του ανθρώπου. Το γεγονός αυτό, σε συνδυασμό με την εξέλιξη του διαδικτύου και την έλευση του Web 2.0 είχαν ως αποτέλεσμα τη ραγδαία αύξηση της υπολογιστικής δύναμης αλλά και της πολυπλοκότητας των συστημάτων. Πλέον, όλοι οι τομείς της ανθρώπινης κοινωνίας βρίσκονται σε μία διαδικασία εκσυγχρόνισης, αλλά και ενσωμάτωσης τεχνολογικών μέσων, που έχει ως αποτέλεσμα τη μέγιστη δυνατή διασύνδεση και τη βελτιστοποίηση της διαχείρισης πόρων. Νέες τεχνικές ανταλλαγής δεδομένων όπως αυτή των Web API δίνει τη δυνατότητα συλλογής τεράστιου όγκου δεδομένων σε ελάχιστο χρόνο, σε αντίθεση με τεχνικές όπως το web scraping. Έτσι λοιπόν, είναι κατανοητό πως υπάρχει μία πρωτοφανής αύξηση κατά την παραγωγή και την αποθήκευση δεδομένων. Γι' αυτό άλλωστε ο 21ος αιώνας χαρακτηρίζεται και ως, ο αιώνας της πληροφορίας. Το μέγεθος όμως, του τεράστιου αυτού όγκου δεδομένων τον καθιστά μη διαχειρίσιμο και ως εκ τούτου μη αξιοποιήσιμο. Κρίνεται ιδιαίτερα σημαντική λοιπόν, η εξέταση της διαδικασίας εξόρυξης χρήσιμης πληροφορίας από μεγάλου όγκου δεδομένα.

1.2 Στόχος - Σκοπός

Κατά τη διαδικασία της εξόρυξης γνώσης από δεδομένα, εκτός από τις τεχνικές μηχανικής μάθησης αλλά και τεχνητής νοημοσύνης, σημαντικό ρόλο αποτελεί η ύπαρξη ενός επιμελητή - παρατηρητή που έχει τη δυνατότητα να διακρίνει διάφορα πρότυπα, πιο άμεσα, λόγω του ανθρώπινου παράγοντα. Μέσα σε ένα τέτοιο πλαίσιο, ο στόχος της παρούσας πτυχιακής εργασίας είναι η εξέταση της εξόρυξης γνώσης από δεδομένα, στο χώρο των Μέσω

Μαζικής Μεταφοράς. Πιο συγκεκριμένα, είναι η προσπάθεια εξαγωγής χρήσιμων συμπερασμάτων σχετικά με την τιμή των εισιτηρίων στο χώρο των αερομεταφορών.

Ο σκοπός της εργασίας αποτελεί τη σχεδίαση και την ανάπτυξη ενός συστήματος που θα συλλέγει δεδομένα σχετικά με την τιμή αεροπορικών εισιτηρίων, θα τα αναλύει και στη συνέχεια θα διαχέει τα αποτελέσματα της έρευνας στο ευρύ κοινό μέσω ενός Web API. Το σύνολο των δεδομένων που θα συλλέγει η εφαρμογή θα προέρχονται από τη βάση δεδομένων της Aegean Airlines.

1.3 Διάρθρωση της Εργασίας

Η εν λόγω πτυχιακή εργασία διαρθρώνεται σε έξι κεφάλαια, τα οποία έχουν ως στόχο την περαιτέρω ανάλυση των σκελών της. Πιο συγκεκριμένα το Κεφάλαιο 1 αποτελεί μια εισαγωγή στο θέμα που πραγματεύεται η εργασία, ενώ παράλληλα αναλύεται ο σκοπός και ο στόχος αυτής. Στη συνέχεια, στο Κεφάλαιο 2 αναλύεται το τεχνικό υπόβαθρο, δηλαδή όλες οι τεχνικές και θεωρητικές γνώσεις που απαιτούνται για την κατανόηση της εργασίας. Έπειτα, ακολουθεί το Κεφάλαιο 3, στο οποίο περιγράφονται οι επιλογές που πραγματοποιήθηκαν σε σχέση με την αρχιτεκτονική σχεδίαση του προτεινόμενου συστήματος καθώς και τα απαραίτητα σχεδιαγράμματα. Συγκεκριμένα αναφέρεται στο κομμάτι της βάσης δεδομένων, των υπηρεσιών και του API. Κατά το επόμενο Κεφάλαιο 4, αναλύεται ο τρόπος υλοποίησης του προτεινόμενου συστήματος, ενώ παράλληλα παρουσιάζεται και επεξηγείται ο κώδικας που αναπτύχθηκε μέσω των κατάλληλων αποσπασμάτων. Στη συνέχεια, το Κεφάλαιο 5 αναφέρεται στο ερευνητικό σκέλος της εργασίας, αφού παρουσιάζει τα πειράματα που πραγματοποιήθηκαν, καθώς και τα συμπεράσματα που εξάχθηκαν από αυτά. Τέλος, στο Κεφάλαιο 6 συνοψίζονται τα κεντρικά συμπεράσματα της εργασίας, ενώ παράλληλα περιγράφονται οι περιορισμοί της αλλά και οι μελλοντικές επεκτάσεις της.

2 Τεχνολογικό Υπόβαθρο

2.1 Εξόρυξη Γνώσης από Δεδομένα

Ο όρος εξόρυξη γνώσης από δεδομένα αναφέρεται στην Ανακάλυψη Γνώσης από Βάσεις Δεδομένων ή αλλιώς Knowledge Discovery in Databases (KDD). Το επιστημονικό αυτό πεδίο ασχολείται με την ανάπτυξη θεωρημάτων και τεχνικών για την ανακάλυψη χρήσιμης πληροφορίας μέσα από μεγάλες συλλογές δεδομένων. Πιο συγκεκριμένα, υπάρχουν πολλοί ορισμοί, όπως αυτός των Frawley, Piatetsky-Shapiro και Matheus (1992) [1] που αναφέρει πως η "Ανακάλυψη Γνώσης από Βάσεις Δεδομένων είναι η ντετερμινιστική διαδικασία εξόρυξης μη αναγνωρίσιμης, άγνωστης και ενδεχομένως χρήσιμης πληροφορίας από δεδομένα". Έτσι λοιπόν, το πεδίο αυτό πραγματεύεται την εξεύρεση κατανοητών προτύπων που μπορούν να ερμηνευτούν και να παράγουν ενδιαφέρουσα πληροφορία. Για την εκπλήρωση αυτού του στόχου χρησιμοποιούνται τεχνικές από διάφορα άλλα επιστημονικά πεδία, όπως αυτό της τεχνητής νοημοσύνης, της στατιστικής και της μηχανικής μάθησης. Συχνά ο όρος εξόρυξη γνώσης από δεδομένα συγχέεται με τον όρο της εξόρυξης δεδομένων. Η εξόρυξη δεδομένων αναφέρεται στη διαδικασία κατά την οποία αναλύονται δεδομένα με τη χρήση διάφορων τεχνικών όπως αλγορίθμων ομαδοποίησης και κατηγοριοποίησης. Η διαδικασία αυτή αποτελεί ένα μόνο βήμα της εξόρυξης γνώσης από δεδομένα. Πιο αναλυτικά, τα βήματα αυτά είναι [2]:

1. **Selection:** Σε αυτό το βήμα επιλέγεται μία συλλογή ή ένα υποσύνολο δεδομένων σύμφωνα με το αντικείμενο της έρευνας
2. **Preprocessing:** Στη συνέχεια πραγματοποιείται μία προεπεξεργασία των δεδομένων για την απομάκρυνση εσφαλμένων ή ελλιπών εγγραφών
3. **Transformation:** Έπειτα τα δεδομένα αυτά μετατρέπονται σε μία κοινή μορφή, ώστε να είναι κατάλληλα για επεξεργασία
4. **Data mining:** Σε αυτό το βήμα εφαρμόζεται η τεχνική εξόρυξης γνώσης που έχει επιλεγεί

5. Interpretation/Evaluation: Τέλος, ακολουθεί η ερμηνεία και η αποτίμηση των αποτελεσμάτων που έχουν προκύψει από το προηγούμενο βήμα

Η παραπάνω επαναληπτική διαδικασία δίνει τη δυνατότητα στους ερευνητές να εξάγουν νέα αλλά και αξιοποιήσιμη πληροφορία από διάφορες συλλογές δεδομένων. Με το τρόπο αυτό η εξόρυξη γνώσης από δεδομένα έχει μεγάλη επίδραση σε διαφορετικούς τομείς, όπως αυτός των Μέσων Μαζικής Μεταφοράς. Πιο συγκεκριμένα οι Branda, Marozzo και Talia, [3] μελέτησαν την ηλεκτρονική συμπεριφορά πελατών, κατά τη διαδικασία αγοράς εισιτηρίων λεωφορείου στην ευρύτερη περιοχή της Καλαβρίας. Χρησιμοποιώντας τεχνικές εξόρυξης διεργασιών συμπέραναν πως υπήρχε αύξηση κατά τους καλοκαιρινούς μήνες, ιδιαίτερα τις ημέρες Σάββατο και Κυριακή ενώ παράλληλα παρατηρήθηκε διαφοροποίηση στη ζήτηση των δρομολογίων.

Σε μία άλλη περίπτωση [4] εξετάστηκε η διακύμανση της τιμής αεροπορικών εισιτηρίων για τα δρομολόγια Los Angeles (LAX) προς Boston (BOS) και Seattle (SEA) προς Washington, DC (IAD). Η συλλογή των δεδομένων που επιλέχτηκε περιείχε 12,000 εγγραφές, εξέταζε έξι διαφορετικές αεροπορικές εταιρείες και πραγματοποιήθηκε σε διάστημα 41 ημερών. Σε αυτή την έρευνα, παρατηρήθηκε πως η ελάχιστη τιμή των αεροπορικών εισιτηρίων εναλλάσσεται κατά μέσο όρο 6 φορές την ημέρα ενώ επίσης πως υπάρχει μεγάλη διακύμανση στις τιμές των ταξιδιών με επιστροφή.

Οι ερευνητές Groves και Gini [5] προσπάθησαν να εξάγουν διάφορα μοτίβα για τη διακύμανση της τιμής αεροπορικών εισιτηρίων, συλλέγοντας 23 εκατομμύρια εγγραφές, σε 109 ημέρες. Το σύνολο των δεδομένων προερχόταν από επτά διαφορετικές αεροπορικές εταιρείες και βασιζόταν σε εγχώρια δρομολόγια στις ΗΠΑ. Υπήρξαν λοιπόν, ενδείξεις για μία κυκλική συμπεριφορά του τιμολογιακού μοντέλου αλλά και μια ισχυρή συσχέτιση με την αύξηση της τιμής του εισιτηρίου και τις ημέρες Σάββατο και Κυριακή.

Τέλος, σύμφωνα με τον Tim Janseen [6] η τιμή ενός αεροπορικού εισιτηρίου συνήθως αυξάνεται καθώς πλησιάζει η μέρα αναχώρησης. Παράλληλα, υπάρχει μία μεγάλη διαφορά την ελάχιστη και στη μέγιστη τιμή. Ο ερευνητής εξέτασε την ύπαρξη προτύπων σε μία συλλογή 126,412 εγγραφών σχετικά με τη συμπεριφορά της τιμής αεροπορικών εισιτηρίων, για το δρομολόγιο μεταξύ των αεροδρομίων San Francisco Airport (SFO) και John F. Kennedy (JFK).

Σύμφωνα με τα παραπάνω, η αξιοποίηση της εξόρυξης γνώσης από δεδομένα αποτελεί έναν αποτελεσματικό τρόπο για την εξαγωγή χρήσιμων συμπερασμάτων σχετικά με τα εξεταζόμενα δεδομένα. Η διαφοροποίηση της παρούσας εργασίας έγκειται στο γεγονός πως η

συλλογή δεδομένων που βρίσκεται υπό εξέταση αναφέρεται σε δρομολόγια που σχετίζονται με ελληνικές πόλεις. Αποτελεί δηλαδή, μία ευκαιρία για τη μελέτη της συμπεριφοράς της τιμής αεροπορικών εισιτηρίων στον ελληνικό χώρο.

2.2 Συλλογή Δεδομένων

Το ερευνητικό μέρος της παρούσας εργασίας καθιστά αναγκαία τη συλλογή δεδομένων σχετικά με την τιμή των εισιτηρίων, στο χώρο των Μέσων Μαζικής Μεταφοράς. Με σκοπό λοιπόν, τη συλλογή τέτοιου είδους δεδομένων, επιλέχτηκε ως πηγή η αεροπορική εταιρεία Aegean Airlines. Η επιλογή αυτή μας επιτρέπει να συλλέξουμε δεδομένα στον τομέα των αερομεταφορών και συγκεκριμένα, δεδομένα σχετικά με την τιμή των αεροπορικών εισιτηρίων.

Η Aegean ιδρύθηκε το 1999, αποτελεί τη μεγαλύτερη ελληνική αεροπορική εταιρεία και είναι μέλος της Star Alliance. Εξυπηρετεί 131 προορισμούς ανά το κόσμο ενώ δραστηριοποιείται κυρίως με πτήσεις εσωτερικού, δηλαδή εγχώριους προορισμούς στην Ελλάδα. Τα τελευταία χρόνια λόγω της επεκτατικής της πολιτικής, είναι σε θέση να εξυπηρετεί τις διασημότερες πόλεις παγκοσμίως αλλά και να μεταφέρει 15 εκατομμύρια επιβάτες ετησίως [7].

Η εταιρεία, μέσω της ιστοσελίδας της, προσφέρει την υπηρεσία Low Calendar Fare, η οποία κατείχε κεντρικό ρόλο κατά τη συλλογή των αεροπορικών δεδομένων. Η υπηρεσία αυτή δίνει τη δυνατότητα στο χρήστη να αναζητήσει αεροπορικά εισιτήρια σε βάθος χρόνου 11 μηνών, για τους προορισμούς που τον ενδιαφέρουν. Έτσι, οι χρήστες είναι σε θέση να βρουν τα πιο οικονομικά αεροπορικά εισιτήρια τόσο για εγχώριους όσο και για εξωτερικούς προορισμούς. Κατά τη χρήση της εν λόγω υπηρεσίας, ο εξυπηρετητής της αεροπορικής εταιρείας αποστέλλει τα δεδομένα σε μορφή JSON. Το μορφότυπο JSON αποτελεί έναν τρόπο αποθήκευσης και αναπαράστασης δεδομένων που χρησιμοποιείται κατά την επικοινωνία δύο συστημάτων [8]. Με αυτό το τρόπο, είμαστε σε θέση να συλλέξουμε τα δεδομένα που είναι αναγκαία για το ερευνητικό σκέλος της εργασίας. Το παρακάτω απόσπασμα κώδικα 2.1 παρουσιάζει τα δεδομένα ενός εισιτηρίου για το δρομολόγιο Αθήνα - Θεσσαλονίκη, όπως αυτά έχουν αποσταλεί από τον εξυπηρετητή.

```

{
  "Inbound":null,
  "Outbound":[
    {
      "Date":"\\\\"/Date(1630368000000)\\\\"/",
      "FullPrice":50.99,
      "Price":50.99,
      "Class":"Economy",
      "Difference":9.00,
      "Error":null,
      "Updated":"\\\\"/Date(1627862400000)\\\\"/",
      "ServiceFee":0.0
    }
  ],
  "InboundLowestPrice":null,
  "OutboundLowestPrice":41.99,
  "CurrencySymbol":"€"
}

```

Απόσπασμα Κώδικα 2.1: Αρχική δομή δεδομένων Aegean

2.3 Βάση Δεδομένων

2.3.1 Επιλογή Τύπου Βάσης Δεδομένων

Μία βάση δεδομένων αποτελεί μία συλλογή από δεδομένα, τα οποία είναι οργανωμένα με τέτοιο τρόπο, έτσι ώστε, να επιτρέπεται η εύκολη ανάκτηση, διαχείριση αλλά και προσπέλαση αυτών. Για τη βέλτιστη διαχείριση τους χρησιμοποιούνται διάφορα Συστήματα Διαχείρισης Βάσεων Δεδομένων (ΣΔΒΔ). Οι βάσεις δεδομένων κατέχουν κεντρικό ρόλο στη βιωσιμότητα των μοντέρνων συστημάτων, καθώς αποτελούν ένα αναπόσπαστο κομμάτι του κύκλου λειτουργίας τους. Στο παρελθόν, έχουν αναπτυχθεί διάφορα μοντέλα βάσεων δεδομένων όπως είναι το ιεραρχικό μοντέλο και το μοντέλο δικτύου, τα οποία όμως δεν αποδείχθηκαν τόσο αποδοτικά. Πλέον, ένα από τα πιο διαδεδομένα μοντέλα βάσεων δεδομένων είναι το σχεσιακό [9].

Κατά το σχεσιακό μοντέλο, οι βάσεις δεδομένων αποκαλούνται σχεσιακές βάσεις δεδομένων και είναι οργανωμένες σύμφωνα με τις λογικές συσχετίσεις των δεδομένων τους. Πιο συγκεκριμένα, τα κύρια συστατικά μιας βάσης δεδομένων είναι οι οντότητες αλλά και οι συσχετίσεις που τις περιγράφουν. Το σύνολο αυτών των στοιχείων παρουσιάζονται με τη

μορφή πινάκων και σχέσεων αντίστοιχα. Μία οντότητα διαθέτει διάφορα γνωρίσματα, τα οποία μπορούν να πάρουν το ρόλο κλειδιού. Σε μία σχεσιακή βάση δεδομένων υπάρχουν πολλά είδη κλειδιών με τα πιο σημαντικά να είναι το πρωτεύον και το ξένο κλειδί. Το πρωτεύον κλειδί είναι μοναδικό για κάθε οντότητα, δεν μπορεί να πάρει την τιμή μηδέν και χαρακτηρίζει μοναδικά μία εγγραφή. Το ξένο κλειδί αναφέρεται σε ένα γνώρισμα που αποτελεί πρωτεύον κλειδί σε μία οντότητα και χρησιμοποιείται για τη δημιουργία συσχετίσεων μεταξύ των οντοτήτων [10]. Για τη διαχείριση σχεσιακών βάσεων δεδομένων χρησιμοποιούνται διάφορα Συστήματα Διαχείρισης Σχεσιακών Βάσεων Δεδομένων γνωστά και ως RDBMS, ενώ για την ανάπτυξη σε αυτά χρησιμοποιείται η γλώσσα SQL. Μερικά από τα πιο δημοφιλή RDBMS είναι η MySQL, η PostgreSQL, η MariaDB καθώς και η Oracle.

Εκτός από τις σχεσιακές βάσεις δεδομένων, υπάρχουν και οι μη σχεσιακές. Οι μη σχεσιακές βάσεις δεδομένων ή αλλιώς και NoSQL, είναι μία σχετικά νέα τεχνολογία που έχει αποκτήσει απήχηση λόγω της εξέλιξης της τεχνολογίας, αλλά και της ανάγκης για τη βέλτιστη διαχείριση μεγάλου όγκου δεδομένων. Τέτοιες βάσεις δεδομένων, δε βασίζονται στις συσχετίσεις μεταξύ των στοιχείων που αποθηκεύουν όπως οι σχεσιακές. Διαθέτουν ελαστική δομή και δεν απαιτούν τη δημιουργία κάποιου σχήματος [11]. Πιο συγκεκριμένα, υπάρχουν πολλά είδη μη σχεσιακών βάσεων δεδομένων, όπου το κάθε ένα από αυτά αποθηκεύει και διαχειρίζεται τα δεδομένα με διαφορετικό τρόπο. Μερικές από τις πιο σημαντικές κατηγορίες μη σχεσιακών βάσεων δεδομένων είναι οι ακόλουθες [12]:

Document

Τέτοιες βάσεις δεδομένων αποθηκεύουν τα δεδομένα σε αρχεία, με τη μορφή ζευγαριών key-value, ενώ χρησιμοποιούν κάποιο είδος μορφότυπου όπως JSON (JavaScript Object Notation) ή XML (eXtensible Markup Language). Μερικές από τις γνωστές document βάσεις δεδομένων είναι η MongoDB αλλά και η CouchDB.

Wide-Column

Σε αυτή την περίπτωση τα δεδομένα αποθηκεύονται σε πίνακες, σειρές και δυναμικές στήλες. Παράλληλα, κάθε σειρά έχει τη δυνατότητα να περιέχει διαφορετικές στήλες. Τέτοιες βάσεις δεδομένων είναι η Cassandra καθώς και η HBase και αξιοποιούνται συνήθως για την αποθήκευση μεγάλου όγκου δεδομένων.

Key-Value

Οι Key-Value βάσεις δεδομένων χρησιμοποιούν αριθμητικά κλειδιά που αναφέρονται σε συγκεκριμένες τιμές. Η μεγάλη διαφορά τους με της Document βάσεις δεδομένων είναι ότι η προσπέλαση των δεδομένων πραγματοποιείται μόνο μέσω των κλειδιών. Μερικές από αυτές τις βάσεις δεδομένων είναι η DynamoDB και η MemcachedB.

Graph

Τέλος, οι βάσεις δεδομένων γράφων, όπως περιγράφει και το όνομα τους, αποθηκεύουν τα δεδομένα με κόμβους γράφου και τις σχέσεις μεταξύ αυτών ως συνδέσεις. Χρησιμοποιούνται κυρίως για την ανάλυση των σχέσεων μεταξύ των δεδομένων. Η OrientDB και η Neo4J αποτελούν βάσεις δεδομένων γράφων.

Σύμφωνα με τα παραπάνω, το μη σχεσιακό μοντέλο διαθέτει αρκετά προτερήματα αλλά χρησιμοποιείται μόνο σε συγκεκριμένες περιπτώσεις. Έτσι, για την ανάπτυξη της προτεινόμενης εφαρμογής επιλέχτηκε το σχεσιακό μοντέλο βάσεων δεδομένων, αφού πληροί καλύτερα τις λειτουργικές ανάγκες του συστήματος. Πιο συγκεκριμένα, το μοντέλο αυτό, προσφέρει μία απλή δομή, ταχύτατη ανάκτηση δεδομένων καθώς και την ακεραιότητα αυτών. Επίσης, ένας από τους σημαντικούς ρόλους για την επιλογή αυτή, ήταν οι δυνατότητες επεξεργασίας και διαχείρισης δεδομένων που παρέχει η γλώσσα SQL.

2.3.1 MySQL

Η MySQL αποτελεί ένα από τα πιο δημοφιλή Συστήματα Διαχείρισης Σχεσιακών Βάσεων Δεδομένων (RDBMS), καθώς χρησιμοποιείται από εταιρείες όπως η Apple, η Netflix και η IBM [13]. Υποστηρίζει όλα τα δημοφιλή λειτουργικά συστήματα, δηλαδή τα Windows, macOS και Linux. Η MySQL αναπτύχθηκε χρησιμοποιώντας τις γλώσσες προγραμματισμού C και C++ από τη σουηδική εταιρεία MySQL AB και η πρώτη ευρέως διαθέσιμη έκδοση της ήταν η 3.11.1 το 1996. Ο κύριος στόχος των δημιουργών της, Michael Widenius, David Axmark και Allan Larsson ήταν να παρέχουν αποδοτική και αξιόπιστη διαχείριση δεδομένων σε ευρεία κλίμακα. Η εταιρεία τους, στη συνέχεια εξαγοράστηκε από τη Sun Microsystems, ενώ πλέον είναι μέρος της Oracle [14]. Η MySQL αποτελεί λογισμικό ανοικτού κώδικα, αφού το 2000 κυκλοφόρησε με την άδεια GNU General Public License (GPL). Παράλληλα κυκλοφορεί και ως εμπορικό προϊόν της Oracle [15].

Η MySQL χρησιμοποιεί το μοντέλο πελάτη/εξυπηρετητή, κατά το οποίο ο πελάτης επικοινωνεί με κάποιον εξυπηρετητή, χρησιμοποιώντας τη γλώσσα SQL (Structured Query Language), με σκοπό την αλληλεπίδραση του με τη βάση δεδομένων [14]. Πιο συγκεκριμένα, μία ή περισσότερες βάσεις δεδομένων φιλοξενούνται από τον εξυπηρετητή. Επομένως, ο πελάτης στέλνει ένα σύνολο εντολών στον εξυπηρετητή, αυτός με τη σειρά του τις εκτελεί και απαντά με μία απόκριση. Η γλώσσα SQL χρησιμοποιείται για την προσπέλαση, την ενημέρωση αλλά και τη διαχείριση των δεδομένων που είναι αποθηκευμένα στη βάση δεδομένων. Μερικά χαρακτηριστικά της MySQL που την αναδεικνύουν σε σχέση με τους ανταγωνιστές της, αποτελούν η αυξημένη συμβατότητα με ένα μεγάλο αριθμό συστημάτων, η υψηλή απόδοσή της αλλά και η ευκολία εγκατάστασης και χρήσης της. Με την άνοδο και την εξέλιξη του cloud computing, η MySQL προσφέρεται και ως διαδικτυακή υπηρεσία, δηλαδή ως Database as a Service (DBaaS). Μερικές από τις εταιρείες που παρέχουν τέτοιες υπηρεσίες είναι η Amazon [16], η Azure [17] και η Heroku [18].

2.4 JavaScript

Η JavaScript αποτελεί μία υψηλού επιπέδου γλώσσα προγραμματισμού σεναρίων ή απλώς scripting language, που συμμορφώνεται με το σύνολο κανόνων της ECMAScript. Η γλώσσα δίνει τη δυνατότητα στο χρήστη να αναπτύξει λογισμικό τόσο για το κομμάτι που εκτελείται από τον περιηγητή όσο και για το κομμάτι που εκτελείται από τον διακομιστή [19]. Μία από τις πιο σημαντικές εκδόσεις της γλώσσας για το σύγχρονο τρόπο ανάπτυξης, ήταν η ECMAScript6, όπου πρόσθεσε σημαντικές λειτουργίες όπως τις promises, τις arrow functions και τα modules [20]. Οι λειτουργίες αυτές συνεισέφεραν στην πορεία της γλώσσας προς τον ασύγχρονο προγραμματισμό.

Η JavaScript δημιουργήθηκε από τον Brendan Eich το 1995 στη Netscape Communications καθώς προοριζόταν για το πρόγραμμα περιήγησης της εταιρείας, Netscape Navigator [21]. Αρχικά, είχε την επωνυμία Mocha, στη συνέχεια αυτή μετατράπηκε σε LiveScript και τέλος, έπειτα από τη συνεργασία της Netscape με τη Sun, σε JavaScript. Ο στόχος των δύο εταιρειών ήταν να προωθήσουν τη JavaScript ως συμπληρωματική, της γλώσσας προγραμματισμού Java. Παρά από μερικές επιφανειακές συντακτικές ομοιότητες, ωστόσο, η JavaScript δε σχετίζεται καθόλου με τη γλώσσα προγραμματισμού Java [22].

Μετά την κυκλοφορία της άρχισε να γίνεται ευρέως διαδεδομένη, αφού όλο και περισσότερα προγράμματα περιήγησης άρχισαν να την υποστηρίζουν. Η κύρια χρήση της

αποτελούσε η ανάπτυξη ιστοσελίδων και εφαρμογών στο πλαίσιο αυτών των προγραμμάτων. Η δυναμική της αλλά και η ευκολία ως προς την εκμάθηση της ήταν οι κύριοι παράγοντες που συνέβαλλαν στη ραγδαία διάδοσή της. Πλέον, πρόκειται για μία από τις πιο δημοφιλείς γλώσσες προγραμματισμού καθώς χρησιμοποιείται από το 68.62% των προγραμματιστών [23]. Ένα ακόμα σημαντικό σημείο καμπής για τη διάδοση της JavaScript αποτέλεσε η δημιουργία του Chrome V8 Engine από την Google καθώς και της πλατφόρμας ανάπτυξης λογισμικού Node.js από τον Ryan Dahl [24]. Με τη βοήθεια των δύο αυτών τεχνολογικών εξελίξεων η JavaScript ήταν πιο αποδοτική ενώ πλέον οι χρήστες ήταν σε θέση να αναπτύξουν λογισμικό που δε θα περιοριζόταν στα προγράμματα περιήγησης.

Promises

Ο ασύγχρονος προγραμματισμός κατέχει κεντρικό ρόλο κατά την ανάπτυξη διαδικτυακών εφαρμογών. Σε ένα τέτοιο πλαίσιο, ένα σύστημα δε περιορίζεται στη γραμμική εκτέλεση εντολών, καθώς του δίνεται η δυνατότητα να εκτελέσει παράλληλα τις εντολές αυτές. Με αυτό το τρόπο, το σύστημα γίνεται πιο αποδοτικό, αφού, ελαχιστοποιείται ο χρόνος απόκρισης του.

Όπως προαναφέρθηκε, η εισαγωγή των promises με την έκδοση ECMAScript6 αποτέλεσε καθοριστικό σημείο για την εξέλιξη της γλώσσας JavaScript. Όλο και περισσότεροι προγραμματιστές εγκατέλειπαν την έως τότε πρακτική διαχείρισης ασύγχρονου κώδικα, δηλαδή τη χρήση συμβάντων, για την πιο νέα και βέλτιστη τεχνική. Κατά την τεχνική αυτή, μία promise αποτελεί μία υπόσχεση για τη διαθεσιμότητα ενός πόρου σε μετέπειτα χρονική περίοδο. Έτσι οι ασύγχρονες συναρτήσεις έχουν τη δυνατότητα να επιστρέφουν τιμές που δεν είναι διαθέσιμες εκείνη τη χρονική στιγμή, σε μετέπειτα στάδιο, με τη χρήση μίας promise. Μία τέτοια promise μπορεί να βρίσκεται στις εξής παρακάτω καταστάσεις [25]:

- **Pending:** Η αρχική κατάσταση για κάθε promise κατά την οποία δεν έχει πραγματοποιηθεί η εκτέλεση του ασύγχρονου κώδικα και η promise δε φέρει κάποια τιμή.
- **Fulfilled:** Σε αυτή την κατάσταση η εκτέλεση του ασύγχρονου κώδικα έχει πραγματοποιηθεί με επιτυχία και η promise έχει αποθηκευμένη την τιμή του αποτελέσματος.

- **Rejected:** Η κατάσταση κατά την οποία η εκτέλεση του ασύγχρονου κώδικα ήταν ανεπιτυχής και η promise έχει την τιμή ενός σφάλματος.

Ένας από τους αρχικούς τρόπους διαχείρισης των promises ήταν μέσω του then statement. Με την τεχνική αυτή μπορεί κανείς να καλεί συναρτήσεις αλυσιδωτά ώστε να διαχειριστεί τα δεδομένα που προκύπτουν από ασύγχρονες συναρτήσεις. Παρ' όλα αυτά, δεν υπήρχε κάποιος εδραιωμένος και βέλτιστος τρόπος εφαρμογής αυτής της τεχνικής. Αποτέλεσμα αυτού, ήταν το φαινόμενο promise hell, στο οποίο η αλυσίδα των συναρτήσεων με τη χρήση του then statement μεγάλωνε διαρκώς [26]. Έτσι ο παραγόμενος κώδικας δεν ήταν πια φιλικός προς τον προγραμματιστή αφού δεν ήταν ευανάγνωστος. Τη λύση για το πρόβλημα αυτό έδωσε η εισαγωγή της λειτουργίας Async/Await.

```
connectDatabase()
  .then((database) => {
    return findAllBooks(database)
      .then((books) => {
        return getCurrentUser(database)
          .then((user) => {
            return pickTopRecommendation(books, user);
          });
      });
  });
```

Απόσπασμα Κώδικα 2.2: Παράδειγμα promise hell[27]

Async / Await

Η λειτουργία του Async/Await πρωτοεμφανίστηκε στη JavaScript το 2018 στην έκδοση ES2017 της ECMAScript [28]. Η προσθήκη της λειτουργίας αυτής, είχε ως στόχο τη βελτίωση της διαχείρισης των promises και κατά συνέπεια του ασύγχρονου προγραμματισμού, αντικαθιστώντας τα then statements. Κατά την τεχνική Async/Await οι ασύγχρονες συναρτήσεις ορίζονται ως συναρτήσεις async με τη χρήση της αντίστοιχης λέξης ενώ επιστρέφουν πάντοτε μία promise. Ο ρόλος της λέξης await είναι η διακοπή της εκτέλεσης της ασύγχρονης συνάρτησης μέχρις ότου μία άλλη ασύγχρονη συνάρτηση να ολοκληρώσει την εκτέλεση της και να επιστρέψει τη δική της promise. Η χρήση του await πραγματοποιείται πάντοτε μέσα σε μία async συνάρτηση [29]. Με το τρόπο αυτό ο κώδικας γίνεται πιο ευανάγνωστος σε σχέση με προηγούμενες μεθόδους, ενώ ο χρήστης είναι σε θέση να

διαχειρίζεται ασύγχρονα κομμάτια κώδικα με μεγαλύτερη ευκολία αλλά και αποτελεσματικότητα.

```
async function f() {  
  
  let promise = new Promise((resolve, reject) => {  
    setTimeout(() => resolve("done!"), 1000)  
  });  
  
  let result = await promise; // wait until the promise resolves (*)  
  
  alert(result); // "done!"  
}  
  
f();
```

***Απόσπασμα Κώδικα 2.3:** Παράδειγμα χρήσης του async/await [30]*

2.5 Node.js

Το Node.js ή αλλιώς Node αποτελεί ένα περιβάλλον εκτέλεσης, ανοιχτού κώδικα που επιτρέπει τη χρήση της JavaScript από τη μεριά του εξυπηρετητή. Χρησιμοποιεί ένα μοντέλο που βασίζεται σε γεγονότα (event driven model) ενώ παράλληλα δεν εμποδίζει την εκτέλεση των εντολών εισόδου/εξόδου [31]. Το ασύγχρονο αυτό πλαίσιο καθιστά το Node.js πολύ αποδοτικό, έχοντας τη δυνατότητα να διαχειριστεί μεγάλο αριθμό ταυτόχρονων συνδέσεων. Έτσι, είναι κατάλληλο για εφαρμογές που παράγουν μεγάλο φόρτο δεδομένων ή απαιτούν μεγάλο αριθμό εντολών εισόδου/εξόδου όπως εφαρμογές IoT (Internet of Things) και εφαρμογές ανταλλαγής μηνυμάτων πραγματικού χρόνου. Μερικές από τις εταιρείες που χρησιμοποιούν το Node.js είναι η Netflix [32] αλλά και η Uber [33], γεγονός που δείχνει την ευρεία διάδοσή του.

Το Node.js είναι βασισμένο στη μηχανή της Google, Chrome V8 ενώ χρησιμοποιεί την libuv βιβλιοθήκη [34] καθώς και τους ασύγχρονους βοηθούς του λειτουργικού συστήματος. Η Chrome V8 Engine είναι ανεπτυγμένη με τη γλώσσα προγραμματισμού C++ και οι αρμοδιότητες της είναι η λειτουργία της ως μεταγλωττιστής, η εκτέλεση του JavaScript κώδικα και η διαχείριση της σειράς εκτέλεσης των συναρτήσεων, γνωστή και ως call stack. Παράλληλα είναι υπεύθυνη για τη διαχείριση των περιττών πόρων (Garbage collection) ενώ παρέχει και τους τύπους δεδομένων, τα ειδικά σύμβολα, τα αντικείμενα και τις συναρτήσεις.

Τέλος, παρέχει ένα event loop ώστε να εκτελείται ασύγχρονα ο κώδικας [35]. Η βιβλιοθήκη libuv εστιάζει κυρίως στην ασύγχρονη είσοδο/έξοδο ενώ παράλληλα προσφέρει ένα σύνολο από νήματα εκτέλεσης (threadpool) [34].

2.5.1 NPM

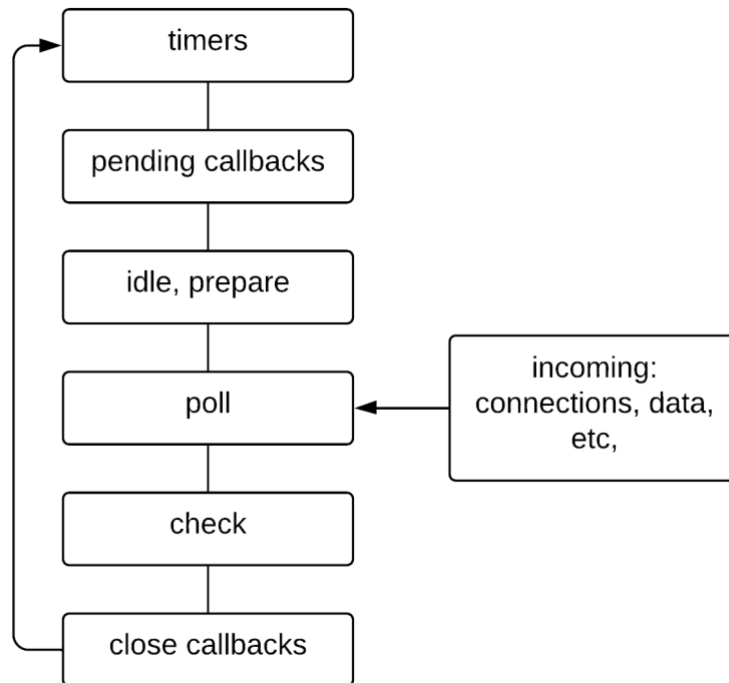
Το NPM η αλλιώς Node Package Manager είναι ο διαχειριστής πακέτων του Node.js. Τα πακέτα αυτά είναι συνήθως ένα σύνολο από περισσότερα modules ή ενότητες που περιγράφονται κεντρικά σε ένα αρχείο με όνομα package.json. Στο αρχείο αυτό υπάρχουν όλα τα πακέτα που χρησιμοποιούνται από το πρόγραμμα καθώς και στοιχεία για αυτά σε σχέση με την έκδοσή τους, το δημιουργό τους, την περιγραφή αλλά και τον τύπο άδειας τους [36]. Ένας από τους πιο συνηθισμένους τρόπους για να εγκαταστήσει κανείς ένα τέτοιο πακέτο στην εφαρμογή του είναι η εκτέλεση της παρακάτω εντολής 2.4, μέσω της γραμμής εντολών [37]. Με το τρόπο αυτό ο χρήστης είναι σε θέση να διαλέξει ανάμεσα σε παραπάνω από ένα εκατομμύριο packages [38] και να τα εγκαταστήσει στην εφαρμογή του.

```
npm install <package_name>
```

Απόσπασμα Κώδικα 2.4: Παράδειγμα εγκατάστασης npm πακέτου

2.5.2 Event Loop

Το event loop του Node.js είναι ένας βρόγχος γεγονότων και αποτελεί ένα από τα πιο σημαντικά κομμάτια λειτουργίας του, δίνοντας τη δυνατότητα εκτέλεσης εντολών εισόδου/εξόδου χωρίς αναμονή. Πιο συγκεκριμένα, διαχειρίζεται την εκτέλεση εντολών όπως τους timers, τα callbacks αλλά και εντολές εισόδου/εξόδου [24]. Κατά την έναρξη του το Node, αρχικοποιεί το βρόγχο γεγονότων, επεξεργάζεται το κώδικα που έχει αναπτυχθεί και στη συνέχεια επιστρέφει για την επεξεργασία του βρόγχου. Η διαδικασία της επεξεργασίας, χωρίζεται σε έξι στάδια τα οποία επαναλαμβάνονται για όσο διάστημα η εφαρμογή εξακολουθεί να έχει κώδικα που πρέπει να εκτελεστεί [39].



Σχήμα 2.1: Στάδια του Event Loop [39]

Σε κάθε στάδιο υπάρχει μία σειρά FIFO (First In, First Out) από callbacks, τα οποία πρέπει να διευθετηθούν. Κάθε callback αποτελεί μία ανάκληση που σηματοδοτεί ένα γεγονός που έχει ολοκληρωθεί. Τα στάδια αυτά είναι τα ακόλουθα [40]:

1. **Timers:** Εκτελούνται όλα τα callbacks που έχουν προγραμματιστεί από τις συναρτήσεις `setTimeout` και `setInterval`.
2. **Pending callbacks:** Εκτελούνται τα callbacks εισόδου/εξόδου που είχαν δημιουργηθεί κατά την προηγούμενη επανάληψη.
3. **Idle, prepare:** Αυτό το στάδιο χρησιμοποιείται εσωτερικά του Node.js
4. **Poll:** Καταγράφονται νέα γεγονότα εισόδου/εξόδου ενώ εκτελούνται και τα αντίστοιχα callbacks.
5. **Check:** Εκτελούνται τα callbacks που έχουν προγραμματιστεί από τη συνάρτηση `setImmediate`
6. **Close callbacks:** Εκτελούνται τα close callbacks, όπως για παράδειγμα ένας τερματισμός μίας σύνδεσης

Με αυτό το τρόπο το Node είναι σε θέση να διαχειριστεί πολλαπλές διεργασίες αποδοτικά, καταναλώνοντας ελάχιστους πόρους.

2.6 Express.js

Η ανάπτυξη λογισμικού στο χώρο του διαδικτύου αποτελεί μια περίπλοκη διαδικασία, καθώς απαιτείται τεχνική γνώση σε διάφορους τομείς αλλά και σωστή διαχείριση πόρων. Για το λόγο αυτό προέκυψε η ανάγκη για την ανάπτυξη εργαλείων που θα απλοποιούσαν τη διαδικασία αυτή. Τα web frameworks αποτελούν ένα σύνολο από εργαλεία και βιβλιοθήκες που ελαχιστοποιούν την προσπάθεια του χρήστη, προσφέροντας συνήθως κάποιο πρότυπο για τη δομή του λογισμικού, ευκολότερη διαχείριση HTTP αιτημάτων καθώς και επιλογές για ασφάλεια και caching [41].

Το Express.js ή αλλιώς Express είναι ένα ελαφρύ Node web framework το οποίο δεν επιβάλλει αυστηρούς κανόνες κατά τον τρόπο ανάπτυξης λογισμικού. Αντίθετα, δίνει τη δυνατότητα στο χρήστη να επιλέξει τη δομή και τα εργαλεία που θα χρησιμοποιήσει. Αποτελεί ένα από τα πιο δημοφιλή Node web frameworks και χρησιμοποιείται από διάσημες εταιρείες όπως είναι η IBM, η Uber αλλά και η Yandex [42]. Το Express προσφέρει στο χρήστη προηγμένες αλλά εύχρηστες τεχνικές routing καθώς και την επιλογή ανάπτυξης middleware λειτουργιών.

Κατά την έναρξη μιας Express εφαρμογής, ο εξυπηρετητής αναμένει για διάφορα HTTP αιτήματα σε προκαθορισμένα από το χρήστη API endpoints. Τα endpoints αυτά, αποτελούν τις διευθύνσεις ή αλλιώς routes του API μέσω των οποίων λειτουργεί το κανάλι επικοινωνίας. Πιο συγκεκριμένα, στην περίπτωση που ο εξυπηρετητής δεχτεί ένα αίτημα HTTP, το καθοδηγεί στο κατάλληλο κομμάτι κώδικα σύμφωνα με το route και τη HTTP μέθοδο, με σκοπό τη διαχείριση του. Μέσω του Express ο χρήστης έχει τη δυνατότητα να ορίσει με ευκολία τη μέθοδο HTTP, το route καθώς και τις middleware συναρτήσεις. Τέτοιες συναρτήσεις έχουν πρόσβαση στο αντικείμενο του αιτήματος (req) αλλά και στην απόκριση (res) καθώς και στην επόμενη middleware συνάρτηση [43].

```
app.get('/', function (req, res) {  
  res.send('Hello World!')  
})  
  
app.post('/', function (req, res) {  
  res.send('Got a POST request')  
})
```

Απόσπασμα Κώδικα 2.5: Παράδειγμα χρήσης Express.js [44]

Το παραπάνω απόσπασμα κώδικα 2.4 περιγράφει ένα απλοϊκό παράδειγμα χρήσης του Express, κατά το οποίο ο χρήστης έχει ορίσει τις middleware συναρτήσεις που θα διαχειριστούν τα HTTP αιτήματα GET και POST, για το ίδιο route.

2.7 Heroku

Πριν από την εποχή των virtual private servers (VPS) οι προγραμματιστές για να αναπτύξουν την εφαρμογή τους στο διαδίκτυο έπρεπε να περάσουν από μία επίπονη και χρονοβόρα διαδικασία. Αρχικά έπρεπε να αγοράσουν ένα φυσικό εξυπηρετητή, να τον διαμορφώσουν κατάλληλα και στη συνέχεια να αναζητήσουν ένα κέντρο δεδομένων, το οποίο θα τους παρείχε φυσικό χώρο, ρεύμα και σύνδεση στο διαδίκτυο [45]. Η εισαγωγή των VPS και του cloud computing απλοποίησε τη διαδικασία αυτή καθώς την άλλαξε ριζικά.

Όπως αναφέρεται από τον οργανισμό τυποποίησης National Institute of Standards and Technology (NIST) [46] το cloud computing αποτελεί την παροχή πόρων όπως δίκτυο, εξυπηρετητές, εφαρμογές και υπηρεσίες μέσω του διαδικτύου, από απομακρυσμένα κεντρικά συστήματα. Το γεγονός αυτό προσφέρει στο χρήστη μεγάλη ευελιξία και υψηλή αυτοματοποίηση, ενώ ταυτόχρονα ελαχιστοποιείται η προσπάθεια του. Σε αυτό το μοντέλο οι χρήστες τιμολογούνται μόνο για την υπολογιστική δύναμη που καταναλώνει το σύστημα τους. Κατά το cloud computing ένα από τα μοντέλα υπηρεσιών είναι και το Platform as a Service (PaaS). Το PaaS δίνει τη δυνατότητα στο χρήστη να αναπτύξει τις εφαρμογές του διαδικτυακά, δίχως να απαιτείται η διαχείριση του δικτύου, των εξυπηρετητών, του λειτουργικού συστήματος ή του χώρου αποθήκευσης.

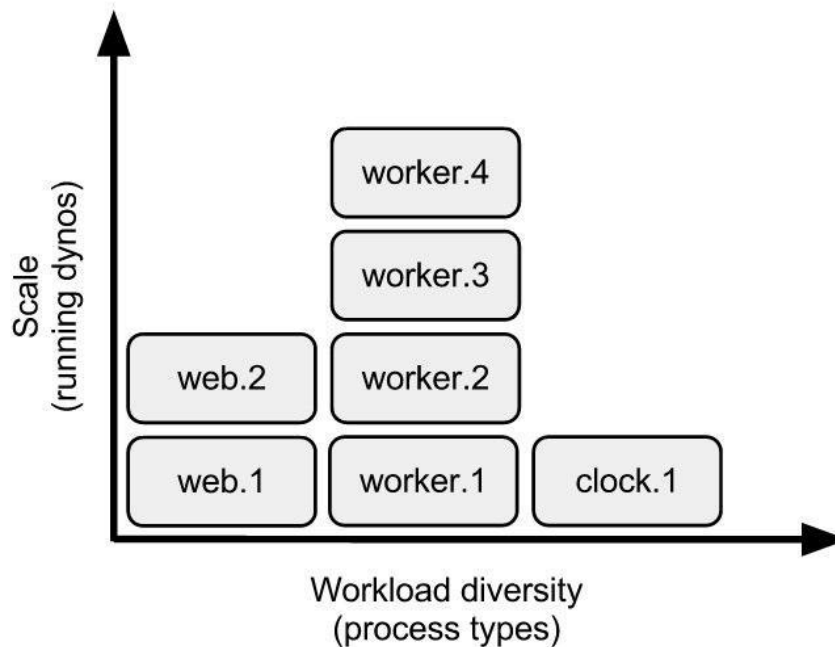
Το Heroku αποτελεί μία τέτοια cloud πλατφόρμα PaaS που δίνει τη δυνατότητα στο χρήστη να αναπτύξει, να διαχειριστεί αλλά και να κλιμακώσει εφαρμογές τελευταίας τεχνολογίας. Δημιουργήθηκε το 2007 από τους James Lindenbaum, Adam Wiggins και Orion Henry ως μία πλατφόρμα που υποστήριζε μόνο εφαρμογές που είναι ανεπτυγμένες με τη γλώσσα προγραμματισμού Ruby [47]. Πλέον το Heroku παρέχει υποστήριξη για οκτώ γλώσσες προγραμματισμού οι οποίες είναι οι Ruby, Java, PHP, Python, Node, Go, Scala και Clojure [48]. Ένα από τα κύρια συστατικά για την ευρεία διάδοση της πλατφόρμας, αποτέλεσε η μεγάλη ποικιλία από επεκτάσεις (add-ons), οι οποίες προσφέρονται μέσω του Heroku Element Marketplace. Έτσι ο χρήστης είναι σε θέση να επεκτείνει αλλά και να βελτιώσει την εφαρμογή του, προσθέτοντας λειτουργίες όπως αυτή της αναζήτησης, του χρονοπρογραμματισμού ή και κάποια βάση δεδομένων.

2.7.1 Heroku Dynos

Τα Heroku Dynos είναι απομονωμένα, εικονικά Linux containers, τα οποία παρέχουν το περιβάλλον για την ανάπτυξη μια εφαρμογής στο Heroku. Η πλατφόρμα προσφέρει διάφορα επίπεδα από dynos ξεκινώντας με το Free επίπεδο, το οποίο διαθέτει 512MB μνήμης RAM ενώ μπορεί να χρησιμοποιηθεί για 550 ώρες [49]. Το παραπάνω επίπεδο αξιοποιήθηκε στην παρούσα εργασία αφού διατίθεται δωρεάν σε όλους τους χρήστες. Μία ακόμη σημαντική διαφορά του Free επιπέδου, είναι ότι τα dynos που παρέχονται, εισέρχονται σε μία κατάσταση αναστολής κατά το διάστημα που δε δέχονται διαδικτυακά αιτήματα. Παράλληλα, υπάρχουν τρεις τύποι από Dynos σε κάθε επίπεδο. Η λειτουργικότητα του καθενός περιγράφεται παρακάτω [50]:

- **Web Dynos:** Χρησιμοποιούνται για τη διαχείριση διαδικτυακών αιτημάτων
- **Worker Dynos:** Χρησιμοποιούνται για την εκτέλεση διάφορων διεργασιών που έχει ορίσει ο χρήστης (συνήθως διεργασίες που εκτελούνται στο παρασκήνιο)
- **One-off Dynos:** Χρησιμοποιούνται για την εκτέλεση μεμονωμένων διεργασιών

Η διαδικασία ενσωμάτωσης μιας εφαρμογής στο Heroku, ξεκινάει επιλέγοντας τον αριθμό αλλά και τον τύπο των dynos που απαιτούνται. Παράλληλα ορίζεται από το χρήστη το σύνολο των διεργασιών που θα εκτελεστούν κατά την έναρξη της, σε ένα αρχείο με το όνομα procfile. Στη συνέχεια ο Dyno Manager, που είναι υπεύθυνος για τη διαχείριση του συνόλου των Dynos, βάζει σε λειτουργία την εφαρμογή εκτελώντας τις διεργασίες αυτές [51].



Σχήμα 2.2: Παράδειγμα dyno formation [52]

Ο Dyno Manager είναι επίσης υπεύθυνος για το κύκλο ζωής των Heroku Dynos. Πιο συγκεκριμένα όλα τα dynos περνάνε από μία διαδικασία ανανέωσης σε καθημερινή βάση, όπου η λειτουργία της εφαρμογής τερματίζει και ξεκινάει εκ νέου. Η διαδικασία αυτή εφαρμόζεται αντίστοιχα από τον Dyno Manager και σε περιπτώσεις όπου προκύπτει κάποιο αναπάντεχο σφάλμα κατά την εκτέλεση μιας εφαρμογής [53].

2.7.2 ClearDB

Όπως προαναφέρθηκε η πλατφόρμα του Heroku δίνει τη δυνατότητα στον χρήστη να επεκτείνει την εφαρμογή του μέσω διαφόρων επεκτάσεων (add-ons). Μία τέτοια επέκταση είναι και η ClearDB που αξιοποιείται ως database as a service (DBaaS).

Πιο συγκεκριμένα, σύμφωνα με την IBM [54], μία database as a service αποτελεί μία cloud computing υπηρεσία, κατά την οποία δίνεται η δυνατότητα πρόσβασης και χρήσης μιας απομακρυσμένης βάσης δεδομένων. Με το τρόπο αυτό δεν απαιτείται από το χρήστη η αγορά και η διαμόρφωση εξοπλισμού ή η διαχείριση και εγκατάσταση του απαραίτητου λογισμικού. Μέσω της ClearDB λοιπόν, ο χρήστης είναι σε θέση να διαμορφώσει τη δική του βάση δεδομένων και να την προσαρμόσει στην εφαρμογή του.

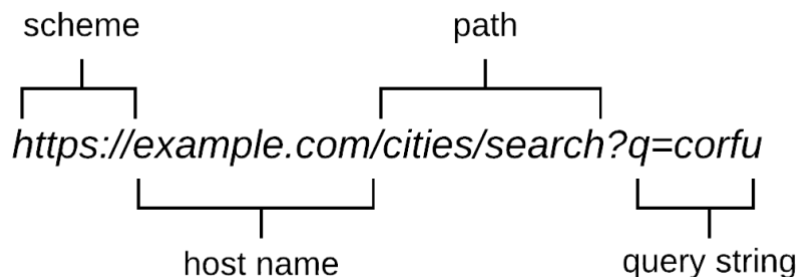
Η ClearDB έχει αναπτυχθεί με στόχο την πρόληψη έκτακτων καταστάσεων κατά τις οποίες η σύνδεση με τη βάση δεδομένων διακόπτεται και διαφθείρεται όγκος δεδομένων. Για

το λόγο αυτό, βρίσκονται σε λειτουργία διάφορα αντίγραφα συμπλεγμάτων βάσεων δεδομένων, ανά το κόσμο. Τα συμπλέγματα αυτά χρησιμοποιούν ένα υβριδικό μοντέλο τεχνολογιών συνδυάζοντας την τεχνική master-master της MySQL καθώς και ένα ειδικό σύστημα SQL routing που βοηθά στην επανασύνδεση των εξυπηρετητών κατά τον εντοπισμό σφαλμάτων [55].

2.8 HTTP

Το HyperText Transfer Protocol ή αλλιώς HTTP δημιουργήθηκε από τον Tim Berners-Lee και την ομάδα του το 1990 και αποτελεί το θεμελιώδες πρωτόκολλο επικοινωνίας στον παγκόσμιο ιστό. Το πρωτόκολλο αυτό, ανήκει στο επίπεδο εφαρμογών του OSI μοντέλου αφού αναπτύσσεται μέσω TCP ή TLS κρυπτογραφημένων συνδέσεων [56]. Χρησιμοποιείται για τη μεταφορά οποιασδήποτε μορφής δεδομένων και λειτουργεί σε ένα πλαίσιο αίτησης-απόκρισης. Πιο συγκεκριμένα, ορίζει τη μορφή της αίτησης που πραγματοποιεί ένας πελάτης (client) προς έναν εξυπηρετητή (server), με σκοπό την ανάκτηση πόρων, καθώς και την αντίστοιχη μορφή της απόκρισης. Οι δύο αυτές μορφές μηνυμάτων απαρτίζονται από μία γραμμή έναρξης, επικεφαλίδες και σώμα.

Σε μία αίτηση, η γραμμή έναρξης περιέχει την HTTP έκδοση, την HTTP μέθοδο αλλά και ένα ενιαίο αναγνωριστικό, γνωστό και ως Uniform Resource Identifier (URI). Ένα URI είναι μία συμβολοσειρά που λειτουργεί ως διεύθυνση ενός πόρου στο διαδίκτυο. Υπάρχουν δύο είδη από URI, το Uniform Resource Locator (URL) που είναι το πιο διαδεδομένο, αλλά και το Uniform Resource Name (URN) [57]. Στο παρακάτω σχήμα 2.3 παρουσιάζονται τα δομικά στοιχεία ενός URL.



Σχήμα 2.3: Δομικά στοιχεία του Uniform Resource Locator (URL)

Στην περίπτωση της απόκρισης, η γραμμή κατάστασης περιέχει την HTTP έκδοση, έναν τριψήφιο κωδικό κατάστασης καθώς και ένα περιγραφικό κείμενο για τον κωδικό αυτό. Ο κωδικός κατάστασης αναφέρεται στην επιτυχημένη ή αποτυχημένη διαχείριση της αίτησης, ενώ το αρχικό του ψηφίο αντιπροσωπεύει το τύπο της απόκρισης. Οι πέντε κατηγορίες κωδικών κατάστασης είναι [58]:

1. 1xx - ενημερωτική απόκριση
2. 2xx - επιτυχία
3. 3xx - ανακατεύθυνση
4. 4xx - σφάλμα πελάτη
5. 5xx - σφάλμα διακομιστή

2.8.1 Μέθοδοι HTTP

Οι HTTP μέθοδοι βρίσκονται όπως προαναφέρθηκε στη γραμμή έναρξης κάθε αίτησης και προσδιορίζουν τον τρόπο με τον οποίο πρέπει να γίνει η διαχείριση από τη μεριά του εξυπηρετητή. Οι πιο διαδεδομένες μέθοδοι είναι οι GET, POST, PUT και DELETE. Στην ενότητα αυτή περιγράφονται οι σημαντικότεροι τύποι των μεθόδων HTTP όπως έχει οριστεί από το HTTP/1.1 [56] [59].

GET

Η μέθοδος GET χρησιμοποιείται για την ανάκτηση συγκεκριμένων πόρων που προσδιορίζονται από το URI. Η μέθοδος αυτή αποκαλείται ασφαλής αφού δεν πρέπει να πραγματοποιούνται αλλαγές από το διακομιστή. Παράλληλα η GET πρέπει είναι ταυτοδύναμη (idempotent), δηλαδή να παράγεται το ίδιο αποτέλεσμα κάθε φορά που πραγματοποιείται η ίδια αίτηση. Στη συνέχεια παρουσιάζεται μια μορφή GET αίτησης.

```
GET http://example.com/cities/corfu.html HTTP/1.1
```

Απόσπασμα Κώδικα 2.6: Παράδειγμα GET αίτησης

HEAD

Η μέθοδος HEAD είναι παρόμοια με τη μέθοδο GET και χρησιμοποιείται μόνο για την ανάκτηση των μεταδεδομένων της επικεφαλίδας ενός πόρου. Αποτελεί και αυτή μία ασφαλής

και ταυτοδύναμη μέθοδο ενώ χρησιμοποιείται συνήθως για τον έλεγχο εγκυρότητας, προσβασιμότητας ή τροποποίησης ενός URL.

POST

Η μέθοδος POST είναι μία από τις πιο σημαντικές μεθόδους, αφού δίνει τη δυνατότητα αποστολής δεδομένων από τον πελάτη (client) προς τον διακομιστή (server). Τέτοια δεδομένα μπορούν για παράδειγμα να είναι το όνομα χρήστη και ο κωδικός πρόσβασης σε κατάλληλη μορφή, κατά τη διαδικασία εισαγωγής σε ένα σύστημα. Ο διακομιστής επεξεργάζεται τα δεδομένα αυτά και πραγματοποιεί κάποιες ενέργειες όπως η εισαγωγή σε μία βάση δεδομένων.

PUT

Η PUT μέθοδος είναι παρόμοια με την POST καθώς αποστέλλονται και με αυτή δεδομένα από τον πελάτη προς τον διακομιστή. Κατά την PUT, ο πόρος που έχει προσδιοριστεί δημιουργείται ή αντικαθίσταται, ανάλογα με τη διαθεσιμότητα του. Μία από τις διαφορές της PUT με την POST είναι ότι η PUT αποτελεί ταυτοδύναμη μέθοδο.

PATCH

Η μέθοδος PATCH εισήχθη στο πρωτόκολλο HTTP το 2010 σε μεταγενέστερο στάδιο και χρησιμοποιείται για τη μερική τροποποίηση ενός πόρου. Η κύρια διαφορά της με την PUT είναι ότι αυτή τροποποιεί τον πόρο ενώ η PUT τον αντικαθιστά.

DELETE

Τέλος, η μέθοδος DELETE αξιοποιείται για τη διαγραφή του πόρου που προσδιορίζεται στην αίτηση. Παράλληλα αποτελεί και αυτή ταυτοδύναμη μέθοδο.

2.8.2 Επικεφαλίδες HTTP

Το HTTP πρωτόκολλο είναι ένα stateless πρωτόκολλο, δηλαδή δε διατηρεί κατάσταση (state) κατά την αμφίδρομη επικοινωνία του πελάτη (client) με τον διακομιστή (server). Πιο συγκεκριμένα, οι αιτήσεις και οι αποκρίσεις που πραγματοποιούνται σε μία σύνδεση δε σχετίζονται μεταξύ τους. Η χρήση των HTTP επικεφαλίδων αντιμετωπίζει αυτή την κατάσταση αφού επιτρέπει τον πελάτη (client) και τον διακομιστή (server) να μοιραστούν πρόσθετες πληροφορίες [60].

Οι επικεφαλίδες αποτελούν μέρος τόσο των αιτήσεων αλλά και των αποκρίσεων. Υπάρχουν τέσσερα είδη από επικεφαλίδες και ο ρόλος τους είναι να παρέχουν χρήσιμες πληροφορίες. Αρχικά, οι επικεφαλίδες γενικού τύπου είναι κοινές για τις αιτήσεις και τις αποκρίσεις και αναφέρονται στο μήνυμα που μεταφέρεται, όπως για παράδειγμα οι Date, Connection και Transfer-Encoding. Στη δεύτερη περίπτωση, οι επικεφαλίδες αίτησης παρέχουν πληροφορίες σχετικά με την αίτηση ή τον πελάτη (client). Μερικές από αυτές είναι οι Host, User-Agent, Accept-Language και Accept-Encoding. Στη συνέχεια υπάρχουν οι επικεφαλίδες απόκρισης, όπως είναι οι Server και Set-Cookie που αναφέρονται στην απόκριση ή στον εξυπηρετητή (server). Τέλος, χρησιμοποιούνται και οι επικεφαλίδες οντότητας που παρέχουν πληροφορίες για το σώμα ή στην περίπτωση που αυτό απουσιάζει, για τον πόρο που έχει προσδιοριστεί [56].

2.8.3 RESTful APIs

Η ανάπτυξη συνεχώς πιο περίπλοκων συστημάτων επέφερε την ανάγκη για πιο αποδοτική επικοινωνία μεταξύ εξυπηρετητών και εφαρμογών. Για το λόγο αυτό δημιουργήθηκαν προγραμματιστικές διεπαφές γνωστές και ως APIs ή αλλιώς Application Programming Interfaces. Οι διεπαφές αυτές, δίνουν τη δυνατότητα σύνδεσης αλλά και αλληλεπίδρασης με ένα σύστημα με σκοπό την ανταλλαγή πόρων. Τα APIs που πλαισιώνονται από το κόσμο του παγκόσμιου ιστού ονομάζονται Web APIs και η αλληλεπίδραση αυτή των συστημάτων στο διαδίκτυο πραγματοποιείται μέσω του πρωτοκόλλου HTTP. Υπάρχουν πολλά είδη Web API όπως είναι το XML-RPC, το JSON-RPC, το SOAP και το REST [61] [62].

Η έννοια του REST ή αλλιώς Representational State Transfer αποδόθηκε αρχικά από τον Roy T. Fielding, κατά την εκπόνηση της διδακτορικής του διατριβής. Σύμφωνα με τον Roy T. Fielding [63], ο όρος REST αναφέρεται σε ένα τρόπο αρχιτεκτονικής σχεδίασης συστημάτων που αλληλεπιδρούν μέσω του διαδικτύου. Τα συστήματα που ακολουθούν τις αρχιτεκτονικές αρχές του REST ονομάζονται RESTful. Τα RESTful APIs χρησιμοποιούν συνήθως το HTTP πρωτόκολλο για επικοινωνία καθώς και το μορφότυπο JSON για την ανταλλαγή πόρων. Παράλληλα, υποστηρίζουν λειτουργίες CRUD (δημιουργία, ανάγνωση, ενημέρωση, διαγραφή), οι οποίες αντιστοιχούν στις μεθόδους HTTP (POST, GET, PUT, DELETE). Τέτοια συστήματα χαρακτηρίζονται από υψηλή αποδοτικότητα, επεκτασιμότητα

αλλά και αξιοπιστία [64]. Οι αρχιτεκτονικές αρχές του REST, όπως περιγράφονται από τον Roy T. Fielding, είναι οι παρακάτω [63] :

Πελάτης-Διακομιστής

Κατά τον τρόπο αρχιτεκτονικής σχεδίασης REST, οι διαδικασίες υλοποίησης του πελάτη (client) και του διακομιστή (server) πρέπει να είναι ξεχωριστές. Πιο συγκεκριμένα, ο πελάτης δε θα πρέπει να έχει τη δυνατότητα να τροποποιήσει τον τρόπο λειτουργίας του διακομιστή και το αντίστροφο. Έτσι, κάθε μέρος έχει τη δυνατότητα να βελτιωθεί και να επεκταθεί ανεξάρτητα από το άλλο. Ο περιορισμός αυτός βελτιώνει την ενσωμάτωση της διεπαφής σε διαφορετικές πλατφόρμες.

Stateless

Ο περιορισμός stateless αναφέρεται στο γεγονός ότι το σύστημα δεν πρέπει να διατηρεί κατάσταση (state) κατά την αμφίδρομη επικοινωνία του πελάτη με τον διακομιστή. Κάθε νέα HTTP αίτηση πρέπει να είναι ξεχωριστή και να εμπεριέχει όλες τις χρήσιμες πληροφορίες. Με το τρόπο αυτό ο αποδέκτης θα είναι σε θέση να εκτελέσει τις κατάλληλες ενέργειες δίχως την αποθήκευση πληροφορίας σχετικά με προηγούμενα μηνύματα.

Cacheable

Κατά το REST ο διακομιστής πρέπει να προσδιορίζει στις αποκρίσεις είτε άμεσα είτε έμμεσα, εάν ο πελάτης έχει τη δυνατότητα να αποθηκεύει προσωρινά τα δεδομένα της απόκρισης στην κρυφή μνήμη (cache). Ο στόχος του περιορισμού αυτού, είναι η βελτίωση της απόδοσης, αφού τα προσωρινά αποθηκευμένα δεδομένα θα μπορούν να επαναχρησιμοποιηθούν δίχως την ανάγκη αποστολής περαιτέρω αιτημάτων.

Ομοιόμορφη Διεπαφή

Ένα RESTful σύστημα θα πρέπει να διαθέτει μία γενική και ομοιόμορφη διεπαφή ανεξάρτητα από την αρχιτεκτονική που διαθέτει ο πελάτης. Πιο συγκεκριμένα, ο τρόπος με τον οποίο πραγματοποιείται η ανάκτηση ενός πόρου θα πρέπει να είναι ίδιος για όλα τα συστήματα. Δηλαδή, θα πρέπει να υπάρχει μία μόνο μορφή αίτησης για κάθε πόρο. Η επίτευξη αυτής της αρχιτεκτονικής επιλογής γίνεται με τον προσδιορισμό τεσσάρων περιορισμών. Αρχικά, ο κάθε πόρος θα πρέπει να αναγνωρίζεται μέσω ενός URI. Στη συνέχεια ένας πελάτης θα πρέπει να είναι σε θέση να εκτελέσει τις απαραίτητες ενέργειες έχοντας μόνο την αναπαράσταση ενός πόρου. Επίσης, οι αιτήσεις και οι αποκρίσεις θα πρέπει να διαθέτουν όλες τις πληροφορίες για

τη διαχείριση τους. Τέλος, κατά την ανάκτηση ενός πόρου, θα πρέπει να παρέχεται μία λίστα από υπερσυνδέσμους που θα αντιστοιχούν σε όλους τους διαθέσιμους πόρους.

Πολυεπίπεδο Σύστημα

Ένα σύστημα θα πρέπει να είναι πολυεπίπεδο, δηλαδή να αποτελείται από πολλά και απομονωμένα επίπεδα που αλληλεπιδρούν μεταξύ τους. Πιο συγκεκριμένα ένα επίπεδο είναι συνήθως υπεύθυνο για μία λειτουργία και μπορεί να επικοινωνήσει μόνο με το επίπεδο που βρίσκεται ακριβώς πάνω ή κάτω από το ίδιο. Με αυτό το τρόπο μειώνεται η περιπλοκότητα του συστήματος αφού διαχωρίζεται αισθητά η κάθε λειτουργία.

Κώδικας κατ' απαίτηση (προαιρετικό)

Ο περιορισμός αυτός είναι προαιρετικός και αναφέρεται στη δυνατότητα του συστήματος να παρέχει κομμάτια εκτελέσιμου κώδικα όπως Java applets ή JavaScript scripts κατ' απαίτηση του πελάτη. Με αυτό το τρόπο ο πελάτης είναι σε θέση να κατεβάσει και να εκτελέσει κομμάτια κώδικα επεκτείνοντας τη λειτουργικότητα του.

3 Σχεδιασμός

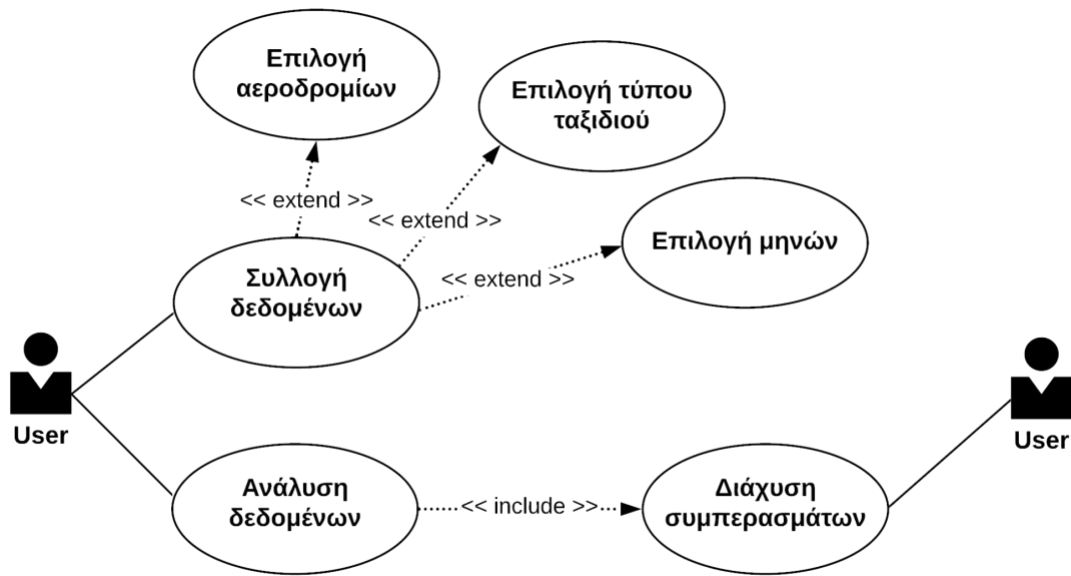
3.1 Καταγραφή απαιτήσεων

Η καταγραφή των απαιτήσεων ενός συστήματος αποτελεί έναν από τους κύριους τομείς κατά την ανάπτυξη λογισμικού. Για τη σωστή ανάλυση τους θα πρέπει να γίνει κατανοητό το τελικό αποτέλεσμα καθώς και ο στόχος του συστήματος. Στην προκειμένη περίπτωση, ενδιαφερόμαστε για την άντληση συμπερασμάτων σχετικά με την τιμή εισιτηρίων Μέσων Μαζικής Μεταφοράς και ειδικότερα, στο χώρο των αερομεταφορών. Ο στόχος της εργασίας αποτελεί την ανάπτυξη μιας διαδικτυακής εφαρμογής, η οποία συλλέγει δεδομένα αεροπορικών εισιτηρίων, τα επεξεργάζεται και τα αναλύει, αντλεί συμπεράσματα και τέλος παρέχει τα συμπεράσματα αυτά στο ευρύ κοινό.

Η προτεινόμενη εφαρμογή υποστηρίζει τις ακόλουθες λειτουργίες:

- Συλλογή δεδομένων εισιτηρίων από ιστοσελίδα εταιρείας μέσων μαζικής μεταφοράς
- Ορισμός αεροδρομίων και δρομολογίων για τη συλλογή των δεδομένων
- Ορισμός τύπου ταξιδιού, με επιστροφή ή χωρίς
- Ορισμός μηνών για τη συλλογή δεδομένων
- Ανάλυση των αποθηκευμένων δεδομένων για την άντληση συμπερασμάτων
- Παροχή των συμπερασμάτων μέσω ενός Web API

Η λειτουργικότητα που περιγράφηκε παραπάνω απεικονίζεται στο ακόλουθο διάγραμμα περιπτώσεων χρήσης (use case diagram).

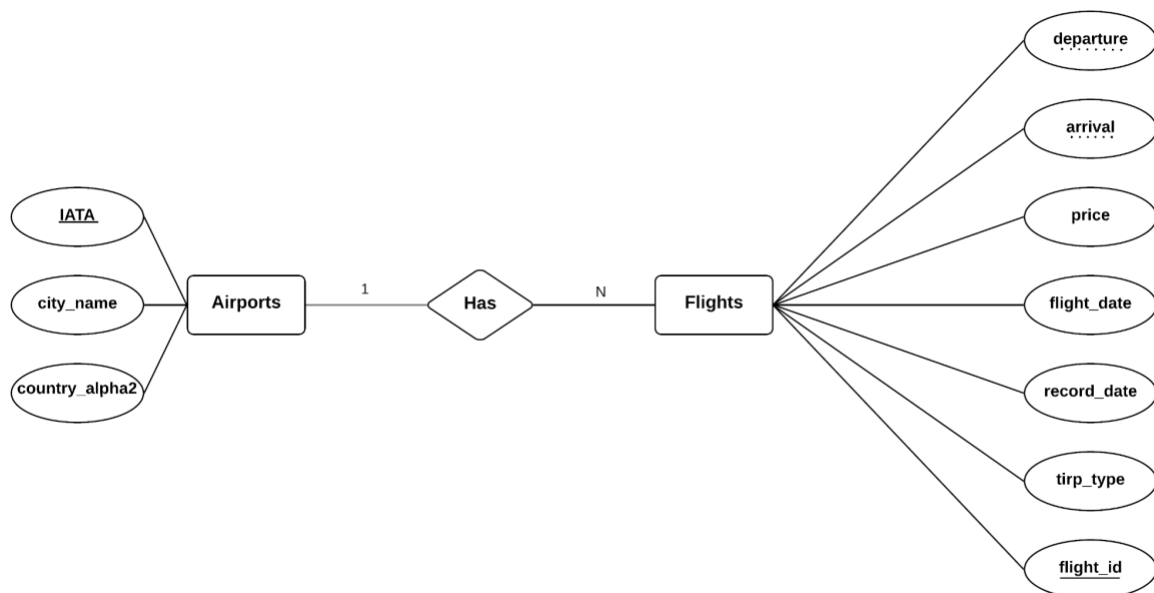


Σχήμα 3.1: Διάγραμμα περιπτώσεων χρήσης

3.2 Σχεδίαση Βάσης Δεδομένων

Η σχεδίαση της βάσης δεδομένων αποτελεί ένα από τα πιο σημαντικά στάδια κατά την ανάπτυξη λογισμικού. Η σωστή υλοποίηση αυτής της διαδικασίας προϋποθέτει την πλήρη κατανόηση και καταγραφή των απαιτήσεων του συστήματος. Για την εργασία αυτή επιλέχθηκε η χρήση μιας σχεσιακής βάσης δεδομένων αφού αντιπροσωπεύει καλύτερα τις ανάγκες της εφαρμογής. Πιο συγκεκριμένα, η επιλογή αυτή, προσφέρει μία απλή δομή, μεγάλη επεκτασιμότητα αλλά και ταχύτατη ανάκτηση δεδομένων. Σε μία τέτοια βάση, τα δεδομένα που συλλέγονται συνήθως αποθηκεύονται σε πίνακες όπου ο κάθε πίνακας αποτελεί μία οντότητα. Η κάθε οντότητα περιγράφεται από ένα σύνολο χαρακτηριστικών τα οποία μπορούν να πάρουν και το ρόλο κλειδιού [10].

Με γνώμονα τις προδιαγραφές του συστήματος, όπως έχουν οριστεί στην προηγούμενη ενότητα 3.1, καθώς και με τα δεδομένα τα οποία αποστέλλονται από τη σελίδα της αεροπορικής εταιρείας, κατασκευάστηκε το διάγραμμα οντοτήτων-συσχετίσεων της βάσης δεδομένων.

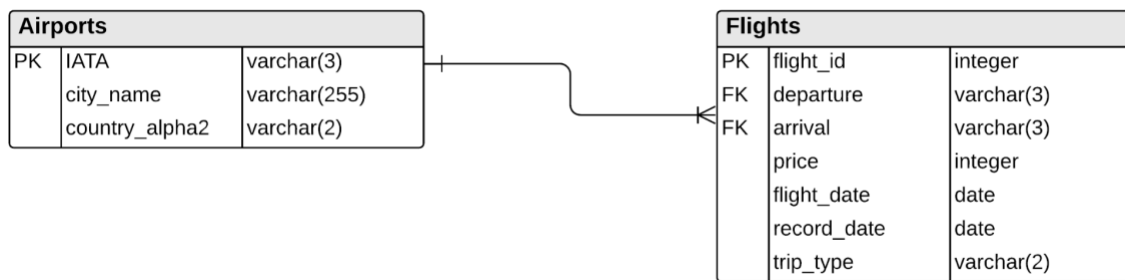


Σχήμα 3.2: Διάγραμμα Οντοτήτων-Συσχετίσεων

Πιο συγκεκριμένα οι οντότητες που το απαρτίζουν είναι, η οντότητα Airports και η οντότητα Flights. Η οντότητα Airports περιγράφει τα αεροδρόμια που χρησιμοποιεί η εφαρμογή για να αντλήσει δεδομένα. Διαθέτει τρία γνωρίσματα τα οποία είναι τα IATA, city_name και country_alpha2. Σε αυτή την οντότητα ως πρωτεύον κλειδί ορίστηκε ο πεδίο IATA. Η οντότητα Flights αναφέρεται στις πτήσεις για τις οποίες συλλέγονται δεδομένα σχετικά με τα αεροπορικά εισιτήρια. Η οντότητα αυτή, κατέχει επτά γνωρίσματα τα οποία είναι τα flight_id, departure, arrival, price, flight_date, record_date και trip_type. Το πεδίο flight_id λειτουργεί ως πρωτεύον κλειδί ενώ πεδία departure, arrival ορίστηκαν ως ξένα κλειδιά που αναφέρονται στο πεδίο city_name της οντότητας Airports. Η συσχέτιση ανάμεσα στις οντότητες είναι η εξής :

Airports - Flights : ένα-προς-πολλά (1:M).

Στο σχήμα 3.3 παρουσιάζεται το σχήμα της βάσης δεδομένων.



Σχήμα 3.3: Σχήμα Crow's Foot

3.2.1 Πίνακας Airports

Ο πίνακας Airports αναφέρεται στα διαθέσιμα αεροδρόμια που μπορεί ο χρήστης να επιλέξει ώστε να αντλήσει δεδομένα εισιτηρίων. Η επιλογή των αεροδρομίων αυτών ορίζει και το εκάστοτε δρομολόγιο.

Airports		
PK	IATA	varchar(3)
	city_name	varchar(255)
	country_alpha2	varchar(2)

Πίνακας 3.1: Πίνακας Airports

Το πεδίο *IATA* εκφράζει τον IATA κωδικό αεροδρομίων, ο οποίος πρόκειται για έναν κωδικό που αποτελείται από τρία γράμματα και έχει θεσπιστεί από τη Διεθνή Ένωση Αερομεταφορών [65]. Ο κωδικός αυτός είναι μοναδικός για κάθε αεροδρόμιο και αντιπροσωπεύει το πρωτεύον κλειδί του πίνακα. Στο πεδίο *city_name* περιγράφεται το όνομα της πόλης στην οποία βρίσκεται το κάθε αεροδρόμιο. Τέλος, το πεδίο *country_alpha2* αποτελεί τον κωδικό χώρας που απαρτίζεται από δύο γράμματα όπως έχει οριστεί από το πρότυπο ISO 3166-1 [66]. Ένας από τους κύριους λόγους για τον τρόπο διαμόρφωσης των πεδίων αποτέλεσε η αυτούσια αποστολή της παραπάνω πληροφορίας από την ιστοσελίδα της αεροπορικής εταιρείας.

3.2.2 Πίνακας Flights

Στο πίνακα με όνομα Flights αποθηκεύονται τα δεδομένα των εισιτηρίων τα οποία έχει συλλέξει και επεξεργαστεί η διαδικτυακή εφαρμογή.

Flights		
PK	flight_id	integer
FK	departure	varchar(3)
FK	arrival	varchar(3)
	price	integer
	flight_date	date
	record_date	date
	trip_type	varchar(2)

Πίνακας 3.2: Πίνακας Flights

Αρχικά το πεδίο *flight_id* αποτελεί έναν μοναδικό για κάθε πτήση αριθμό καθώς και το πρωτεύον κλειδί του πίνακα. Ο αριθμός αυτός αυξάνεται αυτόματα κατά την είσοδο νέων σειρών στη βάση δεδομένων. Στο πεδίο *departure* και *arrival* αποθηκεύονται οι κωδικοί αεροδρομίων IATA από κάθε δρομολόγιο, για τους σταθμούς αναχώρησης και άφιξης αντίστοιχα. Το πεδίο *price* αναφέρεται στην τιμή του εισιτηρίου. Με στόχο την αποθήκευση της ημερομηνίας της πτήσης αλλά και την ημερομηνία κατά την οποία έχει γίνει η λήψη των δεδομένων από το σύστημα μας, έχουν οριστεί δύο πεδία, το *flight_date* και το *record_date*. Τα δύο αυτά πεδία δρουν και ως ξένα κλειδιά, αφού αναφέρονται στον αεροπορικό κωδικό IATA του πίνακα Airports. Τέλος, το πεδίο *trip_type* αναφέρεται στον τύπο του ταξιδιού που έχει επιλέξει ο χρήστης. Πιο συγκεκριμένα υπάρχουν δύο επιλογές, ταξίδι με επιστροφή RT (Round Trip) και ταξίδι χωρίς επιστροφή OW (One Way).

3.3 Αρχιτεκτονική Συστήματος

Η αρχιτεκτονική σχεδίαση ενός συστήματος είναι ίσως η κυριότερη διαδικασία κατά την ανάπτυξη λογισμικού, αφού περιλαμβάνει τον προσδιορισμό των σχέσεων μεταξύ των

δομικών στοιχείων καθώς και τις ιδιότητες αυτών. Αποτελεί μία διαδικασία μέσω της οποίας επιτυγχάνεται η πλήρης κατανόηση του συστήματος ενώ παράλληλα διακρίνεται ο κύριος σκοπός ανάπτυξης του [67]. Για την άρτια διεκπεραίωση της αρχιτεκτονικής σχεδίασης δεν αρκεί κανείς να λάβει υπόψιν του μόνο τις λειτουργικές απαιτήσεις όπως περιγράφηκαν στο σκέλος 3.2.1 αλλά και τα ποιοτικά χαρακτηριστικά που θα διέπουν την εφαρμογή.

Ο πρωταρχικός στόχος της παρούσας εργασίας είναι η ανάπτυξη μιας εφαρμογής, με την οποία ο χρήστης θα έχει τη δυνατότητα, να συλλέξει δεδομένα στον τομέα των Μέσων Μαζικής Μεταφοράς και ειδικότερα των αερομεταφορών, να τα αναλύσει, να εξάγει διάφορα συμπεράσματα και τέλος να παρέχει τα συμπεράσματα αυτά στο ευρύ κοινό, για περαιτέρω αξιοποίηση. Η συλλογή των δεδομένων θα πραγματοποιείται μέσω του εξυπηρετητή της αεροπορικής εταιρείας Aegean Airlines. Παράλληλα, η εφαρμογή αυτή θα πρέπει να είναι ευέλικτη, επεκτάσιμη αλλά και να επιτρέπει εύκολη και αποδοτική συντήρηση. Για το λόγο αυτό επιλέχθηκε μία ξεκάθαρη αρχιτεκτονική μορφή, κατά την οποία οι λειτουργίες της εφαρμογής διαχωρίζονται σε τρεις υπηρεσίες. Πιο συγκεκριμένα οι υπηρεσίες αυτές είναι η υπηρεσία συλλογής δεδομένων, η υπηρεσία ανάλυσης δεδομένων αλλά και η υπηρεσία διάχυσης των στατιστικών στοιχείων και συμπερασμάτων στο ευρύ κοινό μέσω ενός Web API. Στη συνέχεια περιγράφονται οι υπηρεσίες αυτές καθώς και τα αντίστοιχα διαγράμματα ροής δεδομένων.

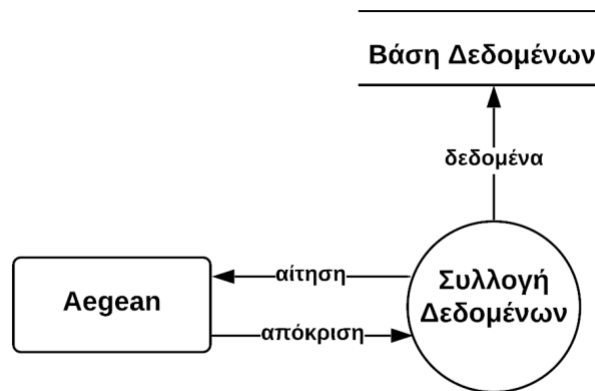


Σχήμα 3.4: Διάγραμμα Ροής Δεδομένων Μηδενικού Επιπέδου

Υπηρεσία Συλλογής Δεδομένων

Η υπηρεσία συλλογής δεδομένων είναι μία από τις πιο κρίσιμες υπηρεσίες του συστήματος αφού είναι υπεύθυνη για τη συλλογή αλλά και την αποθήκευση των δεδομένων στη βάση δεδομένων του συστήματος. Αρχικά, η υπηρεσία επικοινωνεί με τον διακομιστή της αεροπορικής εταιρείας αποστέλλοντας ένα HTTP αίτημα. Στη συνέχεια δέχεται την απόκριση

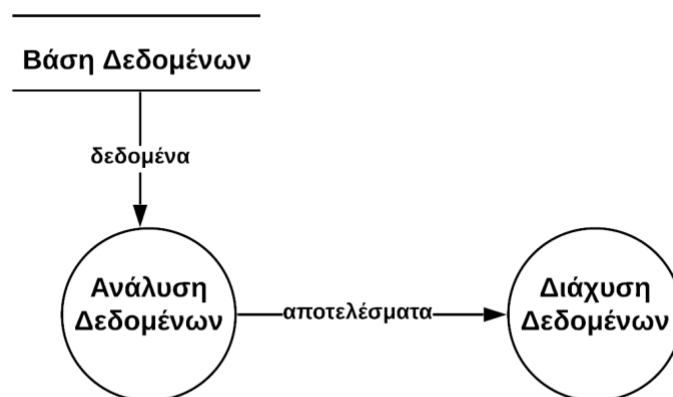
του διακομιστή και την επεξεργάζεται ώστε να συλλεχθούν μόνο τα χρήσιμα δεδομένα. Τέλος, αποθηκεύει τα δεδομένα αυτά στη βάση δεδομένων του συστήματος.



Σχήμα 3.5: Διάγραμμα Ροής Δεδομένων Πρώτου Επιπέδου - Υπηρεσία Συλλογής Δεδομένων

Υπηρεσία Ανάλυσης Δεδομένων

Η υπηρεσία ανάλυσης των δεδομένων δίνει τη δυνατότητα στο χρήστη να αναλύσει τα δεδομένα που έχουν συλλεχθεί, παράγοντας έτσι διάφορα στατιστικά στοιχεία, όπως η μέση τιμή, η διάμεσος, η διακύμανση, το εύρος και η τυπική απόκλιση. Με αυτό τον τρόπο, ο χρήστης είναι σε θέση να συγκρίνει τα αποτελέσματα και στη συνέχεια να πραγματοποιήσει μία ποιοτική ανάλυση για αυτά. Η υπηρεσία, αρχικά αντλεί τα απαιτούμενα δεδομένα από τη βάση δεδομένων, εφόσον αυτά έχουν συλλεχθεί, τα αναλύει και τα παρέχει στο Web API.



Σχήμα 3.6: Διάγραμμα Ροής Δεδομένων Πρώτου Επιπέδου - Υπηρεσία Ανάλυσης Δεδομένων

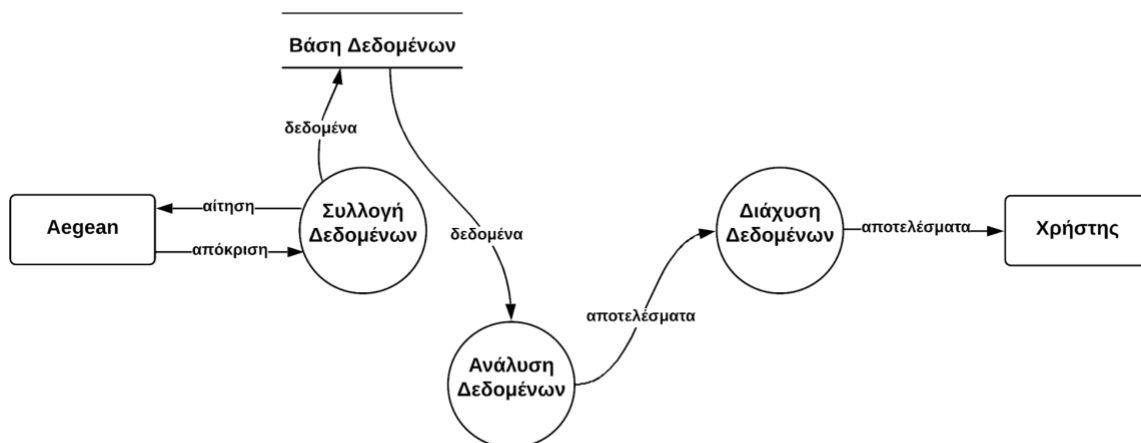
Υπηρεσία Διάχυσης Συμπερασμάτων

Στο πλαίσιο της ανάπτυξης αυτής της εφαρμογής, κρίθηκε αναγκαίος ένας τρόπος με τον οποίο θα παρέχονται τα ευρήματα της έρευνας στο ευρύ κοινό. Δηλαδή, οι εξωτερικοί χρήστες να είναι σε θέση να λάβουν τα αποτελέσματα αλλά και τα συμπεράσματα των ερευνητών ώστε να τα αξιοποιήσουν σε περαιτέρω έρευνες ή εφαρμογές. Για το λόγο αυτό, η υπηρεσία της διάχυσης αποτελείται από ένα Web API, μέσω του οποίου ο ερευνητής έχει τη δυνατότητα να ορίσει τα συμπεράσματα που έχει εξάγει και να τα διαθέσει στο ευρύ κοινό.



Σχήμα 3.7: Διάγραμμα Ροής Δεδομένων Πρώτου Επιπέδου - Υπηρεσία Διάχυσης Συμπερασμάτων

Σύμφωνα με όσα αναφέρθηκαν παραπάνω σχεδιάστηκε το διάγραμμα ροής δεδομένων πρώτου επιπέδου που παρουσιάζει τη ροή της πληροφορίας μέσα στο κύκλο λειτουργίας του συστήματος.

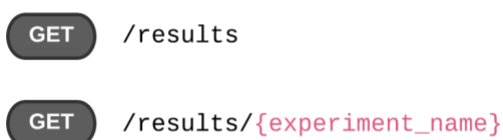


Σχήμα 3.8: Διάγραμμα Ροής Δεδομένων Πρώτου Επιπέδου

3.3.1 Αρχιτεκτονική RESTful API

Η άρτια αρχιτεκτονική σχεδίαση ενός API αποτελεί μία άκρως σημαντική διαδικασία αφού, το κομμάτι της εφαρμογής που αλληλεπιδρά με εξωτερικούς χρήστες, είναι αυτό του RESTful API. Κατά τη διαδικασία αυτή, είναι απαραίτητο να έχει κατανοήσει κανείς όλες τις πτυχές του συστήματος αλλά και τις ανάγκες των εξωτερικών χρηστών που θα το αξιοποιήσουν. Είναι σαφές λοιπόν πως η ευχρηστία, η λιτότητα αλλά και η συνεκτικότητα πρέπει να αποτελούν τους γνώμονες σύμφωνα με τους οποίους πραγματοποιείται η σχεδίαση.

Η διαδικασία αρχιτεκτονικής σχεδίασης του RESTful API ξεκίνησε με τον ορισμό του πλήθους αλλά και του είδους των API endpoints. Πιο συγκεκριμένα το API αποτελείται από δύο endpoints που χρησιμοποιούν την HTTP μέθοδο GET. Δηλαδή, για να έχει κάποιος πρόσβαση στους πόρους που παρέχει το σύστημα, θα πρέπει να πραγματοποιήσει HTTP αιτήματα με τη μέθοδο GET στα κατάλληλα endpoints. Τα endpoints αυτά είναι τα ακόλουθα:



GET /results

GET /results/{experiment_name}

***Σχήμα 3.9:** Αρχιτεκτονική των API endpoints*

Όπως παρουσιάζεται παραπάνω το πρώτο endpoint του API έχει τη διεύθυνση /results. Πραγματοποιώντας το κατάλληλο HTTP αίτημα σε αυτό το μονοπάτι, ο χρήστης θα είναι σε θέση να λάβει ως απόκριση από το διακομιστή του συστήματος, όλες τις πληροφορίες σχετικά με τα ευρήματα της έρευνας. Πιο αναλυτικά, η απόκριση του διακομιστή είναι σε μορφή JSON ενώ τα δεδομένα που παρέχονται στο χρήστη είναι διαχωρισμένα σύμφωνα με τα πειράματα που έχουν οριστεί από τους ερευνητές. Στον πίνακα 3.3 περιγράφονται όλες οι πληροφορίες που παρέχει ο διακομιστής ως απόκριση.

Πεδίο	Περιγραφή
experiments	Το σύνολο των πειραμάτων που έχουν προσδιοριστεί
experiment_name	Το όνομα του πειράματος
description	Περιγραφή του πειράματος
source	Η πηγή από την οποία έχουν συλλεχθεί τα

	δεδομένα
airports	Τα αεροδρόμια του πειράματος
targets	Τα αεροδρόμια που έχουν απομονωθεί και λειτουργούν μόνο ως σημεία άφιξης
trip_type	Ο τύπος ταξιδιού (RT, OW)
request_interval	Το μεσοδιάστημα των HTTP αιτημάτων
start_date	Η ημερομηνία έναρξης του πειράματος
end_date	Η ημερομηνία λήξης του πειράματος
routes	Τα δρομολόγια που έχουν εξεταστεί
departure	Ο κωδικός IATA του αεροδρομίου αναχώρησης
arrival	Ο κωδικός IATA του αεροδρομίου άφιξης
findings	Τα συμπεράσματα που έχουν εξαχθεί έπειτα από τη διαδικασία της ποιοτικής αποτίμησης
results	Τα στατιστικά στοιχεία που έχουν προκύψει κατά τη διαδικασία της ανάλυσης δεδομένων

Πίνακας 3.3: Περιγραφή πεδίων απόκρισης

Στην περίπτωση του δεύτερου endpoint που έχει ως μονοπάτι το `/results/{experiment_name}` ο χρήστης είναι σε θέση να λάβει δεδομένα για ένα μόνο πείραμα που τον ενδιαφέρει. Κατά το δεύτερο σκέλος του μονοπατιού βρίσκεται η μεταβλητή `experiment_name`, μέσω της οποίας ο χρήστης μπορεί να ορίσει το όνομα του πειράματος. Έτσι λοιπόν έχει τη δυνατότητα να απομονώνει συγκεκριμένα πειράματα και να ανακτά περιορισμένη πληροφορία.

Μέσω αυτής της αρχιτεκτονικής σχεδίασης το σύστημα παρέχει τον απαραίτητο όγκο πληροφορίας ενώ παράλληλα παραμένει πιστό στην αρχή της επεκτασιμότητας. Ένα τέτοιο σύστημα θα μπορούσε να θεωρηθεί εύπλαστο αφού οι μελλοντικοί χρήστες έχουν τη δυνατότητα να το τροποποιήσουν σύμφωνα με τις ανάγκες τους.

```
{
  "experiments": [
    {
      "experiment_name": {
```

```

    "description": "This experiment was conducted to ...",
    "source": "AEGEAN",
    "airports": "ATH,SKG",
    "trip_type": "RT",
    "request_interval": "three times a day",
    "start_date": "01-09-2021",
    "end_date": "30-09-2021",
    "routes": [
        {
            "departure": "ATH",
            "arrival": "SKG",
            "results": {
                "mean_price": ...,
                "median_price": ...,
                "range": ...,
                "variance": ...,
                "standard_deviation": ...,
                ...
            }
        },
        {
            "departure": "SKG",
            "arrival": "ATH",
            "results": {
                "mean_price": ...,
                "median_price": ...,
                "range": ...,
                "variance": ...,
                "standard_deviation": ...,
                ...
            }
        },
    ],
    "findings": {
        ...
    }
}
....
]
}

```

Απόσπασμα Κώδικα 3.1: Υποθετικό παράδειγμα απόκρισης για το endpoint /results

4 Υλοποίηση

4.1 Εγκατάσταση εργαλείων

Το πρώτο βήμα κατά την υλοποίηση της προτεινόμενης διαδικτυακής εφαρμογής ήταν η εγκατάσταση των απαιτούμενων εργαλείων. Η ανάπτυξη του λογισμικού έγινε σε τοπικό περιβάλλον, χρησιμοποιώντας το λειτουργικό σύστημα macOS και τη γλώσσα προγραμματισμού JavaScript. Αρχικά πραγματοποιήθηκε η εγκατάσταση του περιβάλλοντος εκτέλεσης Node.js μέσω του macOS Installer, και συγκεκριμένα της έκδοσης v14.15.5. Παράλληλα εγκαταστάθηκε η έκδοση v7.14.0 του διαχειριστή πακέτων του Node, Node Package Manager (NPM). Για να δει κανείς τις εκδόσεις των συστημάτων, που προαναφέρθηκαν, στο δικό του τοπικό περιβάλλον μπορεί να εκτελέσει τις παρακάτω εντολές 4.1.

```
$ node -v  
$ npm -v
```

Απόσπασμα Κώδικα 4.1: Εντολές εγκατάστασης Node.js και npm

Στη συνέχεια πραγματοποιήθηκε η εγκατάσταση της έκδοσης v4.17.1 του Express.js. Όπως περιγράφεται στη συνέχεια το Express χρησιμοποιήθηκε κατά την υλοποίηση του RESTful API. Για την εγκατάσταση του εργαλείου αυτού χρησιμοποιήθηκε η παρακάτω εντολή.

```
$ npm install express --save
```

Απόσπασμα Κώδικα 4.2: Εντολή εγκατάστασης Express.js

Ακολουθεί μία συνοπτική περιγραφή των βιβλιοθηκών που χρησιμοποιήθηκαν κατά την ανάπτυξη του συστήματος.

4.1.1 Βιβλιοθήκες

dotenv

Κατά την ανάπτυξη του προγράμματος κρίθηκε αναγκαία η απόκρυψη των ευαίσθητων στοιχείων της βάσης δεδομένων. Πιο συγκεκριμένα τα στοιχεία αυτά είναι το όνομα του host και του user, ο κωδικός πρόσβασης της βάσης δεδομένων αλλά και το όνομα της. Για την επίτευξη αυτού του στόχου τα στοιχεία αυτά ορίστηκαν ως μεταβλητές περιβάλλοντος (environment variables) όπως ορίζεται από τη μεθοδολογία Twelve Factor App [68]. Στη συνέχεια χρησιμοποιήθηκε η βιβλιοθήκη dotenv για την αξιοποίηση τους. Η εγκατάσταση της πραγματοποιήθηκε με τη χρήση της εντολής 4.3.

```
$ npm install dotenv
```

***Απόσπασμα Κώδικα 4.3:** Εντολή εγκατάστασης του εργαλείου dotenv*

Η βιβλιοθήκη dotenv παρέχει στη διαδικασία process.env, η οποία είναι υπεύθυνη για τη διαχείριση του περιβάλλοντος χρήστη [69], τις μεταβλητές που ορίζει ο χρήστης σε ένα .env αρχείο. Έτσι, δημιουργήθηκε ένα κρυφό αρχείο τύπου .env το οποίο περιέχει όλα τα ευαίσθητα στοιχεία.

```
DB_HOST=localhost  
DB_ADMIN_USERNAME=*****  
DB_ADMIN_PASSWORD=*****  
DB_NAME=flight_data
```

***Απόσπασμα Κώδικα 4.4:** Αρχείο .env*

Έπειτα ενσωματώθηκε στο πρόγραμμα μέσω του Node, με την παρακάτω γραμμή κώδικα.

```
require("dotenv").config();
```

***Απόσπασμα Κώδικα 4.5:** Ενσωμάτωση της βιβλιοθήκης dotenv*

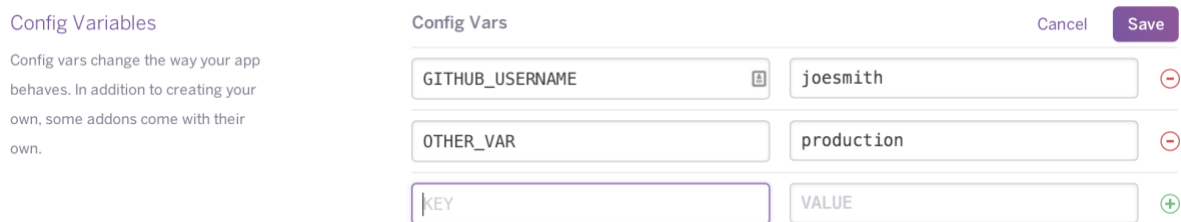
Με αυτό το τρόπο μας δόθηκε η δυνατότητα να αποκρύψουμε τα ευαίσθητα στοιχεία της βάσης δεδομένων και να την προστατέψουμε από διάφορους κακόβουλους χρήστες.

```
var pool = mysql.createPool({  
  host: process.env.DB_HOST,  
  user: process.env.DB_ADMIN_USERNAME,  
  password: process.env.DB_ADMIN_PASSWORD,
```

```
database: process.env.DB_NAME,  
});
```

Απόσπασμα Κώδικα 4.6: Ορισμός ευαίσθητων στοιχείων της βάσης δεδομένων

Κατά το στάδιο ενσωμάτωσης του προγράμματος με το Heroku και της ClearDB όπως περιγράφεται στις επόμενες ενότητες 4.1.2 και 4.2 αντίστοιχα, κρίθηκε αναγκαία η τροποποίηση του τρόπου λειτουργίας του, με στόχο την εκτέλεση του σε online περιβάλλον. Τα ευαίσθητα στοιχεία παρέχονται πλέον από το πάνελ διαχείρισης της ClearDB ενώ προστίθενται στο Heroku Dashboard ως Configuration Variables.



Σχήμα 4.1: Παράδειγμα καταχώρησης configuration variables στο Heroku

node-fetch

Ένα από τα κύρια κομμάτια λειτουργίας της προτεινόμενης διαδικτυακής εφαρμογής αποτελεί η συλλογή δεδομένων από τον εξυπηρετητή μιας αεροπορικής εταιρείας. Η επικοινωνία αυτή με τον εξυπηρετητή επιτυγχάνεται μέσω διάφορων HTTP κλήσεων. Όταν κανείς χρησιμοποιεί τη JavaScript από τη μεριά του πελάτη (client) η λειτουργία αυτή επιτυγχάνεται μέσω του fetch API αλλά και της συνάρτησης window.fetch(). Ένα από τα modules που προσφέρουν αυτή τη λειτουργία στο Node.js είναι και το node-fetch. Η εγκατάσταση του node-fetch πραγματοποιείται μέσω της εντολής 4.7.

```
$ npm install node-fetch
```

Απόσπασμα Κώδικα 4.7: Εντολή εγκατάστασης της βιβλιοθήκης node-fetch

Το node-fetch αποτελεί ένα ελαφρύ module που μας δίνει τη δυνατότητα να αξιοποιήσουμε στο Node.js τη λειτουργικότητα της συνάρτησης window.fetch(), με τη χρήση μιας νέας συνάρτησης που ονομάζεται fetch() [70]. Έτσι κατά την εκτέλεση του προγράμματος πραγματοποιείται ένα πλήθος HTTP κλήσεων τύπου GET για τη συλλογή δεδομένων.

```
let response = await fetch(url);
```

```
let flights = await response.json();
```

Απόσπασμα Κώδικα 4.8: Παράδειγμα χρήσης της βιβλιοθήκης node-fetch

4.1.2 Heroku

Η φύση της παρούσας εργασίας απαιτούσε τη συλλογή και την αποθήκευση δεδομένων καθώς και τη συνεχή αλληλεπίδραση με εξωτερικούς χρήστες, μέσω του Web API. Για το λόγο αυτό ήταν απαραίτητη η εκτέλεση του προγράμματος στο πλαίσιο μιας cloud πλατφόρμας, όπως το Heroku [71]. Έτσι, με την εκτέλεση του προγράμματος σε απομακρυσμένους διακομιστές επιτεύχθηκε η αδιάλειπτη συλλογή δεδομένων καθώς και η αλληλεπίδραση με εξωτερικούς χρήστες.

Με στόχο την ενσωμάτωση του συστήματος στην πλατφόρμα του Heroku, πραγματοποιήθηκε αρχικά η εγκατάσταση του Heroku CLI σε τοπικό περιβάλλον. Το Heroku CLI αποτελεί μία διεπαφή γραμμής εντολών που παρέχεται από το Heroku και χρησιμοποιείται για την αλληλεπίδραση με την πλατφόρμα. Για την εγκατάσταση της αξιοποιήθηκε η παρακάτω εντολή.

```
$ brew install heroku/brew/heroku
```

Απόσπασμα Κώδικα 4.9: Εντολή εγκατάστασης Heroku

Στη συνέχεια πραγματοποιήθηκε η σύνδεση με το Heroku παρέχοντας το όνομα λογαριασμού αλλά και το κωδικό πρόσβασης μέσω της παρακάτω εντολής.

```
$ heroku login -i
```

Απόσπασμα Κώδικα 4.10: Εντολή σύνδεσης με το Heroku

Έπειτα δημιουργήθηκε ένα απομακρυσμένο git αποθετήριο [72]. Το git αποτελεί ένα σύστημα ελέγχου κώδικα ή αλλιώς version control system, το οποίο δίνει τη δυνατότητα παρακολούθησης αλλαγών κατά την ανάπτυξη λογισμικού. Ένα τέτοιο αποθετήριο αποτελεί μία έκδοση του προγράμματος που φιλοξενείται σε κάποιο απομακρυσμένο διακομιστή. Σε αυτή την περίπτωση ο διακομιστής αυτός ανήκει στην πλατφόρμα Heroku.

```
$ heroku create flight-data
```

Απόσπασμα Κώδικα 4.11: Εντολή δημιουργίας απομακρυσμένου αποθετηρίου στο Heroku

Με στόχο τη σωστή εκτέλεση του προγράμματος από τον Dyno Manager του Heroku, δημιουργήθηκε ένα αρχείο τύπου procfile. Μέσα στο αρχείο αυτό προσδιορίστηκε ο τύπος των διεργασιών που θα εκτελέσει ο Dyno Manager καθώς και τα script έναρξης.

```
web: npm run start_server
worker: npm start start_worker
```

Απόσπασμα Κώδικα 4.12: Αρχείο procfile

Τέλος, πραγματοποιήθηκε η έναρξη της εφαρμογής στο Heroku χρησιμοποιώντας τις εντολές του git, commit και push.

4.2 Υλοποίηση Βάσης Δεδομένων

Στο στάδιο της υλοποίησης της βάσης δεδομένων επιλέχτηκε το Σύστημα Διαχείρισης Βάσεων Δεδομένων (ΣΔΒΔ) MySQL [73]. Η MySQL αποτελεί ένα από τα πιο διαδεδομένα ΣΔΒΔ και επιλέχτηκε λόγω του μεγάλου όγκου υλικού και οδηγιών που υπάρχουν στο διαδίκτυο. Αρχικά, η εγκατάσταση της πραγματοποιήθηκε σε τοπικό περιβάλλον και στη συνέχεια έγινε ενσωμάτωση στην πλατφόρμα του Heroku [71] μέσω της επέκτασης ClearDB [55].

Η δημιουργία της βάσης δεδομένων επιτεύχθηκε με χρήση του παρακάτω κώδικα 4.13. Πιο συγκεκριμένα δημιουργήθηκε η βάση δεδομένων, οι πίνακες Airports και Flights καθώς και οι επιμέρους συσχετίσεις μεταξύ τους.

```
CREATE DATABASE FlightData;

CREATE TABLE Airports (
  IATA VARCHAR(3) NOT NULL,
  city_name VARCHAR(255) NOT NULL,
  country_alpha2 VARCHAR(2) NOT NULL,
  PRIMARY KEY (IATA)
);

CREATE TABLE Flights (
  flight_id INTEGER NOT NULL AUTO_INCREMENT,
  departure VARCHAR(3) NOT NULL,
  arrival VARCHAR(3) NOT NULL,
  price INTEGER NOT NULL,
```

```

flight_date DATE NOT NULL,
record_date DATE NOT NULL,
trip_type VARCHAR(2) NOT NULL,
PRIMARY KEY (flight_id)
FOREIGN KEY (departure) REFERENCES Airports (IATA)
FOREIGN KEY (arrival) REFERENCES Airports (IATA)
);

```

Απόσπασμα Κώδικα 4.13: Δημιουργία Βάσης Δεδομένων

Η ενσωμάτωση της βάσης δεδομένων με το Node.js πραγματοποιήθηκε χρησιμοποιώντας το npm πακέτο mysql. Το πακέτο αυτό, είναι ένας Node.js driver ανοιχτού κώδικα με άδεια MIT, για τη MySQL [74], το οποίο δίνει τη δυνατότητα να αλληλεπιδράσουμε με τη βάση δεδομένων, όπως ανοίγοντας διάφορες συνδέσεις και εκτελώντας ερωτήματα SQL.

```

var mysql = require("mysql");
var pool = mysql.createPool({
  host: process.env.DB_HOST,
  user: process.env.DB_ADMIN_USERNAME,
  password: process.env.DB_ADMIN_PASSWORD,
  database: process.env.DB_NAME,
});

```

Απόσπασμα Κώδικα 4.14: Ενσωμάτωση της βιβλιοθήκης mysql

Στο παραπάνω απόσπασμα κώδικα 4.14 εισάγεται το module mysql και στη συνέχεια ορίζονται τα απαιτούμενα πεδία για τη σύνδεση με τη βάση δεδομένων. Τα πεδία αυτά είναι το όνομα του host και του user, ο κωδικός πρόσβασης της βάσης δεδομένων αλλά και το όνομα της. Για την καλύτερη διαχείριση των πόρων χρησιμοποιήθηκε η τεχνική connection pooling [15]. Κατά το connection pooling, το πρόγραμμα αντί να δημιουργεί μία σύνδεση, κάθε φορά που αλληλεπιδρά με τη βάση δεδομένων, διατηρεί ένα πλήθος από ανοικτές συνδέσεις σε ένα σύνολο pool, καθ' όλη τη διάρκεια εκτέλεσης του. Έτσι, όταν διάφορες διεργασίες αλληλεπιδρούν με τη βάση δεδομένων, δε χρειάζεται να δημιουργείται νέα σύνδεση κάθε φορά, παρά μόνο να επαναχρησιμοποιείται μία σύνδεση από το pool. Η τεχνική αυτή είναι κατάλληλη για την εξοικονόμηση των πόρων του συστήματος.

Για την αρχικοποίηση του πίνακα Airports χρησιμοποιήθηκε το παρακάτω απόσπασμα κώδικα 4.14. Πιο συγκεκριμένα πραγματοποιείται μία κλήση στο διακομιστή της αεροπορικής εταιρείας, ο οποίος επιστρέφει δεδομένα για όλα τα ενεργά αεροδρόμια που εξυπηρετεί η εταιρεία. Στη συνέχεια αυτά αποθηκεύονται στη βάση δεδομένων με την κατάλληλη σειρά.

```

let airports = [];

const url = (href =
"https://el.aegeanair.com/sys/flights/airports?_=1617199794254");

request.get(url, (error, response, body) => {
  let json = JSON.parse(body);

  for (let airport in json) {
    airports.push([json[airport]["value"], json[airport]["label"],
json[airport]["country"]]);
  }

  var sql = "INSERT INTO Airports (IATA, city_name, country_alpha2) VALUES ?";

  pool.getConnection(function (err, connection) {
    if (err) {
      errorlog.error(`Error Message : ${err}`);
    }

    let sql = "INSERT INTO Airports (IATA, city_name, country_alpha2) VALUES ?";

    connection.query(sql, [airports], function (error) {
      if (error) {
        errorlog.error(`Error Message : ${error}`);
        reject(error);
      } else {
        connection.release();
        resolve("Task was successful");
      }
    });
  });
});

```

Απόσπασμα Κώδικα 4.15: Αρχικοποίηση του Πίνακα Airports

4.3 Scheduling

Η άντληση συμπερασμάτων από έναν όγκο δεδομένων είναι άμεσα συνδεδεμένη με τη σωστή διαχείριση και συλλογή των δεδομένων αυτών. Έτσι, η επαναληπτική διαδικασία της συλλογής απαιτεί συνέπεια σε βάθος χρόνου. Για την επίτευξη αυτού του στόχου κρίθηκε αναγκαίος ο προγραμματισμός του χρόνου εκτέλεσης του προγράμματος.

Αρχικά, για την εκτέλεση του προγράμματος σε τοπικό περιβάλλον αναπτύχθηκε ένα κομμάτι κώδικα που είναι υπεύθυνο για τον προγραμματισμό και την εκτέλεση συγκεκριμένων λειτουργιών, χρησιμοποιώντας το Node module, node-cron [75]. Η εγκατάσταση του γίνεται όπως οποιοδήποτε άλλο module, με το τρόπο που έχει προαναφερθεί, δηλαδή μέσω του NPM:

```
$ npm install --save node-cron
```

Απόσπασμα Κώδικα 4.16: Εντολή εγκατάστασης node-cron

Στο παρακάτω απόσπασμα κώδικα περιγράφεται η συνάρτηση scheduler καθώς και ένας τρόπος χρήσης της.

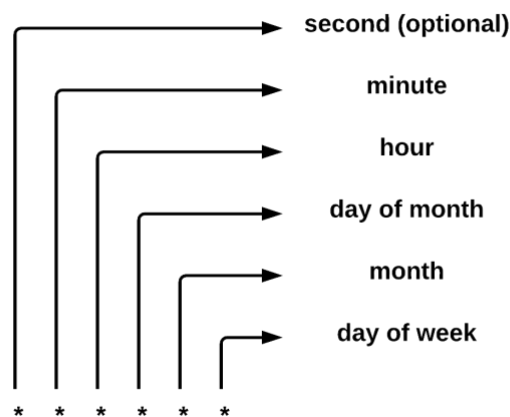
```
var cron = require("node-cron");

/**
 * This function is used only for local deployment
 *
 *
 * Schedules a task to run at the specified interval
 *
 * @param {String} jobInterval - The interval that the task will be run
 * @param {Object} airports - Object containing all of the airports and options for
 * setting airport targets and trip type
 */
function scheduler(jobInterval, airports) {
  cron.schedule(jobInterval, handleFlightData(airports));
}

// An example of usage for running tasks every day, for August
let interval = "* * * Aug *";
airportGroups.forEach((airportGroup) => {
  scheduler(interval, airportGroup);
});
```

Απόσπασμα Κώδικα 4.17: Συνάρτηση scheduler

Η βιβλιοθήκη node-cron δίνει τη δυνατότητα στο χρήστη, να ορίσει το πότε θα εκτελούνται συγκεκριμένες λειτουργίες. Πιο συγκεκριμένα μπορεί να οριστεί ένα μεσοδιάστημα με τη μορφή που παρουσιάζεται στο σχήμα 4.2.



Σχήμα 4.2: Ορισμός μεσοδιαστήματος διεργασίας [75]

Για την κατανόηση της λειτουργικότητας παρατίθεται ένα παράδειγμα.

Έστω τρεις περιπτώσεις μεσοδιαστημάτων.

1. * * * * * : εκτέλεση του κώδικα σε καθημερινή βάση
2. * 5 * * * Sun : εκτέλεση του κώδικα κάθε Κυριακή του μήνα ανά 5 λεπτά
3. * * * * Aug */2 : εκτέλεση του κώδικα κάθε δεύτερη μέρα για το μήνα Αύγουστο

Παρέχοντας λοιπόν ένα μεσοδιάστημα, στη συνάρτηση scheduler καθώς και την απαραίτητη συνάρτηση επιτεύχθηκε η συλλογή δεδομένων, όπως αυτή ορίστηκε κατά τη διαδικασία του πειραματισμού. Η συνάρτηση scheduler όπως περιγράφεται στο απόσπασμα κώδικα 4.17 διατίθεται για περαιτέρω ανάπτυξη της εφαρμογής σε τοπικό περιβάλλον, από μελλοντικούς χρήστες.

Στη δεύτερη φάση κατά την ανάπτυξη της εφαρμογής, κρίθηκε αναγκαία η μεταφορά της υπηρεσίας του χρονοπρογραμματισμού στην πλατφόρμα του Heroku. Η επιλογή αυτή ήταν απαραίτητη λόγω του τρόπου λειτουργίας του Heroku, όπου τα Heroku Dynos ανακυκλώνονται τουλάχιστον μία φορά την ημέρα. Για το λόγο αυτό ενδεικνύεται η χρήση του Heroku Scheduler. Η ενσωμάτωση με τον Heroku Scheduler ήταν αρκετά εύκολη αφού πραγματοποιείται μέσω μιας γραφικής διεπαφής. Έτσι, προσδιορίστηκε το μεσοδιάστημα εκτέλεσης του προγράμματος, καθώς και το αρχείο που είναι υπεύθυνο για τη συλλογή και την αποθήκευση των δεδομένων.

4.4 Διαχείριση Σφαλμάτων

Στην ανάπτυξη λογισμικού τα σφάλματα είναι ένα αναπόφευκτο μέρος της διαδικασίας. Τα σφάλματα αυτά μπορούν διαχωριστούν σε δύο μεγάλες κατηγορίες, τα προγραμματιστικά και τα λειτουργικά σφάλματα [76]. Τα προγραμματιστικά σφάλματα αναφέρονται κυρίως σε σφάλματα που έχουν προκύψει από λάθη προγραμματιστών, όπως για παράδειγμα όταν μία ασύγχρονη συνάρτηση εκτελείται χωρίς να έχει οριστεί μία συνάρτηση ανάκλησης ή όταν μία μεταβλητή έχει την τιμή null και δεν μπορεί να διαβαστεί. Στην αντίθετη περίπτωση τα λειτουργικά σφάλματα περιγράφουν μία προβληματική κατάσταση στην οποία βρίσκεται το σύστημα. Μερικές από αυτές είναι η αδυναμία του συστήματος να συνδεθεί με κάποιο εξυπηρετητή, η ανεπάρκεια μνήμης ή ένα αίτημα που έχει εξαντληθεί χρονικά [77].

Ένας από τους τρόπους έγκαιρης σύλληψης τέτοιων σφαλμάτων είναι η χρήση των try catch blocks. Κατά την τεχνική αυτή, αρχικά, εκτελείται ένα σύνολο εντολών μέσα σε ένα κομμάτι κώδικα try. Στην περίπτωση που δεν προκύψει κάποιο σφάλμα το πρόγραμμα συνεχίζει την εκτέλεση του. Αντίθετα, εκτελείται το κομμάτι κώδικα που βρίσκεται μέσα στο catch block και περιέχει τη διαχείριση του σφάλματος. Υπάρχει και μία ακόμα προαιρετική κατάσταση finally που εκτελείται κατά την ολοκλήρωση των παραπάνω βημάτων [78].

Με στόχο την ορθή γνωστοποίηση σφαλμάτων χρησιμοποιήθηκε η βιβλιοθήκη winston. Η βιβλιοθήκη αυτή δίνει τη δυνατότητα καταγραφής σφαλμάτων σε διάφορες μορφές για την ενημέρωση του χρήστη.

```
/**
 * Logger configuration
 *
 * Used for error messages
 */
const winston = require("winston");

module.exports = winston.createLogger({
  transports: [
    new winston.transports.File({
      name: "error-file",
      level: "error",
      filename: "./logs/error.log",
      json: false,
      datePattern: "yyyy-MM-dd-",
      prepend: true,
    }),
  ],
});
```

```
],  
});
```

Απόσπασμα Κώδικα 4.18: Ενσωμάτωση βιβλιοθήκης winston

Το παραπάνω απόσπασμα κώδικα περιγράφει την αρχικοποίηση της βιβλιοθήκης στο Node.js με σκοπό την εξαγωγή της για αξιοποίηση στο κεντρικό κορμό του προγράμματος. Κατά την καταγραφή ενός σφάλματος γίνεται η εγγραφή του σε ένα αρχείο error.log το οποίο βρίσκεται μέσα στο φάκελο logs. Η νέα αυτή εισαγωγή περιγράφει το είδος του σφάλματος καθώς την ημερομηνία και την ώρα που έχει προκύψει. Με αυτό το τρόπο ο χρήστης είναι σε θέση να ενημερώνεται για την εκτέλεση του προγράμματος. Στο απόσπασμα κώδικα 4.19 παρουσιάζεται ένα παράδειγμα χρήσης του εργαλείου winston.

```
try {  
  let airports = await getAirports(airportGroup);  
  
  let hasTargets = false;  
  if (airportGroup.targets.length !== 0) {  
    hasTargets = true;  
  }  
  
  let flightData = await getFlightData(airports, airportGroup.tripType,  
hasTargets, airportGroup.tripDates);  
  
  await storeData(flightData);  
} catch (error) {  
  errorlog.error(`Error Message : ${error}`);  
}
```

Απόσπασμα Κώδικα 4.19: Παράδειγμα χρήσης βιβλιοθήκης winston

4.5 Συλλογή και Αποθήκευση Δεδομένων

Η συλλογή δεδομένων είναι μία από τις βασικές λειτουργίες της διαδικτυακής εφαρμογής, που αναπτύχθηκε στα πλαίσια αυτής της εργασίας. Λόγο της ερευνητικής της φύσης απαιτείται οι χρήστες να έχουν τη δυνατότητα να συλλέξουν δεδομένα, επιλέγοντας οι ίδιοι τα αεροδρόμια και κατά συνέπεια τα δρομολόγια, για τα οποία θα αντλήσουν συμπεράσματα. Το παρακάτω απόσπασμα κώδικα 4.20 παρουσιάζει τον ορισμό επιλογών από το χρήστη.

```

let airportGroups = [
  {
    airports: "BER,CDG,MAD,FCO,LON,ATH",
    targets: "ATH",
    tripType: "RT",
    tripDates: { departureDate: "2021-9", arrivalDate: "2021-9" },
  },
  {
    airports: "BER,CDG,MAD,FCO,LON,JTR,JMK,CFU,RHO,MLO",
    targets: "JTR,JMK,CFU,RHO,MLO",
    tripType: "RT",
    tripDates: { departureDate: "2021-9", arrivalDate: "2021-9" },
  },
  {
    airports: "ATH,SKG",
    targets: "",
    tripType: "OW",
    tripDates: { departureDate: "2021-9", arrivalDate: "2021-9" },
  },
  {
    airports: "ATH,SKG",
    targets: "",
    tripType: "RT",
    tripDates: { departureDate: "2021-9", arrivalDate: "2021-9" },
  },
];

```

Απόσπασμα Κώδικα 4.20: Αρχικοποίηση airportGroups

Πιο συγκεκριμένα ο κάθε χρήστης μπορεί να ορίσει τις ομάδες αεροδρομίων (airportGroups) για τις οποίες θα συλλέξει δεδομένα. Κάθε ομάδα αεροδρομίων περιέχει μία λίστα από αεροδρόμια, μια σήμανση targets, τον τύπο του ταξιδιού όπως με επιστροφή (RT) ή χωρίς (OW) και τους μήνες αναχώρησης και επιστροφής για τους οποίους θέλει να συλλέξει δεδομένα. Η σήμανση targets δίνει τη δυνατότητα στο χρήστη να απομονώσει κάποια αεροδρόμια κατά τη συλλογή. Για την κατανόηση της λειτουργίας παρατίθεται ένα παράδειγμα.

Έστω μία περίπτωση όπου ο χρήστης επιθυμεί να συλλέξει δεδομένα για τα αεροδρόμια Αθήνα (ATH), Θεσσαλονίκη (SKG), Βερολίνο (BER), Λονδίνο (LON). Στην περίπτωση που το πεδίο targets παραμένει κενό η δημιουργία των δρομολογίων διαμορφώνεται ως εξής:

ATH - SKG

ATH - BER

ATH - LON

SKG - BER

SKG - LON

Στην αντίθετη περίπτωση που ο χρήστης επιθυμεί να απομονώσει το αεροδρόμιο BER, τότε η δημιουργία των δρομολογίων διαμορφώνεται ως εξής:

ATH - BER

SKG - BER

LON - BER

Με αυτό το τρόπο λοιπόν ο χρήστης είναι σε θέση να εξατομικεύσει την επιλογή δρομολογίων.

Κατά την εκτέλεση του προγράμματος αρχικά καλείται η συνάρτηση `processAirportGroups` η οποία δέχεται ως παράμετρο όλες τις ομάδες αεροδρομίων που έχει ορίσει ο χρήστης και στη συνέχεια καλεί τη συνάρτηση `handleFlightData` για κάθε μία από αυτές.

```
processAirportGroups (airportGroups);

/**
 * Processes all of the given airport groups by calling the main function
 * handleFlightData for each group
 *
 * @param {Array<Object>} airportGroups - Object containing all of the airport
 * groups that we want to get data for
 */
async function processAirportGroups (airportGroups) {
  for (const airportGroup of airportGroups) {
    await handleFlightData (airportGroup);
  }
  pool.end();
}
```

Απόσπασμα Κώδικα 4.21: Συνάρτηση `processAirportGroups`

Η συνάρτηση `handleFlightData` αποτελεί τον κορμό του συστήματος καθώς είναι υπεύθυνη για τη διαχείριση των δεδομένων. Πιο συγκεκριμένα, καλεί τη συνάρτηση

getAirports με σκοπό να αποθηκευτούν τα δεδομένα σχετικά με τα αεροδρόμια που έχει επιλέξει ο χρήστης και ελέγχει αν ο χρήστης έχει απομονώσει κάποια αεροδρόμια με την επιλογή targets. Έπειτα καλεί τη συνάρτηση getFlightData ώστε να συλλεχθούν δεδομένα για τα δρομολόγια που έχουν οριστεί. Τέλος, καλεί τη συνάρτηση storeData με στόχο την αποθήκευση των δεδομένων στη βάση. Σε περίπτωση που κάποιο από τα στάδια λειτουργίας αποτύχει καταγράφεται ένα νέο γεγονός σφάλματος στο αρχείο error.log.

```
/**
 * Main function, responsible for handling flight data (Calling the
 * appropriate functions for collecting, parsing and storing)
 *
 * @param {Object} airportGroup - Object containing all of the airports
 * alongside with options for setting airport targets and trip type
 */
async function handleFlightData(airportGroup) {
  try {
    let airports = await getAirports(airportGroup);

    let hasTargets = false;
    if (airportGroup.targets.length !== 0) {
      hasTargets = true;
    }

    let flightData = await getFlightData(airports, airportGroup.tripType,
hasTargets, airportGroup.tripDates);

    await storeData(flightData);
  } catch (error) {
    errorlog.error(`Error Message : ${error}`);
  }
}
```

Απόσπασμα Κώδικα 4.22: Συνάρτηση handleFlightData

Μέσω της συνάρτησης getAirports πραγματοποιείται η ανάκτηση του σύνολου των αεροδρομίων που έχει ορίσει ο χρήστης. Πιο συγκεκριμένα, η συνάρτηση δέχεται ως παράμετρο μία ομάδα αεροδρομίων (airportGroup), συνδέεται στη βάση δεδομένων και ανακτά τα αεροδρόμια που έχουν καθοριστεί. Τέλος, προσδίδει στα αεροδρόμια που έχουν απομονωθεί μία ένδειξη target για τη μετέπειτα διαχείριση τους. Έτσι, επιστρέφεται στη συνάρτηση handleFlightData το σύνολο των απαιτούμενων αεροδρομίων που απαιτούνται για την εκτέλεση του προγράμματος.

```

/**
 * Returns all of the airports as an array of objects
 *
 * @param {Object} airportGroup - Object containing all of the airports alongside
 * with options for setting airport targets and trip type
 * @return {Promise} airportList - Promise array contains all of the airports
 */
function getAirports(airportGroup) {
  return new Promise((resolve, reject) => {
    pool.getConnection(function (err, connection) {
      if (err) {
        throw err;
      }

      let q =
        airportGroup.airports === "ALL"
          ? "SELECT * FROM Airports"
          : "SELECT * FROM Airports WHERE FIND_IN_SET(IATA, ?)";

      connection.query(q, [airportGroup.airports], (error, result) => {
        connection.release();
        if (error) {
          errorlog.error(`Error Message : ${error}`);
        }

        let airportList = JSON.parse(JSON.stringify(result));

        airportList.forEach((airport) => {
          airport["target"] = false;

          if (airportGroup.targets.includes(airport.IATA)) {
            airport["target"] = true;
          }
        });

        return resolve(airportList);
      });
    });
  });
}

```

Απόσπασμα Κώδικα 4.23: Συνάρτηση getAirports

Η συνάρτηση `getFlightData` είναι υπεύθυνη για τη συλλογή των δεδομένων. Αρχικά, δέχεται ως είσοδο τη λίστα των αεροδρομίων που έχει επιλέξει ο χρήστης, τον τύπο του ταξιδιού, μία ένδειξη για το αν ο χρήστης έχει επιλέξει να απομονώσει κάποια αεροδρόμια καθώς και τους μήνες για τους οποίους θέλει να συλλέξει δεδομένα. Στη συνέχεια δημιουργεί τα απαιτούμενα δρομολόγια, ορίζοντας το αεροδρόμιο αναχώρησης και άφιξης καθώς και το αντίστοιχο URL. Τέλος, καλεί τις συναρτήσεις `getFlights` και `addFlights` για κάθε ένα από τα δρομολόγια. Με το τρόπο αυτό συλλέγονται τα δεδομένα και επιστρέφονται στη συνάρτηση `handleFlightData` για περαιτέρω διαχείριση.

```
/**
 * Returns flight data for all of the given airports
 *
 * @param {Array<Object>} airports - Array of all of the airports
 * @param {String} tripType - String that identifies the trip type, either Round
 * Trip (RT) or One Way (OW)
 * @param {String} hasTargets - String that identifies if there are airport targets
 * set
 * @param {Object} tripDates - Object containing departure and arrival dates
 * @return {Promise} flightDataResult - Promise array contains all of the flight
 * info
 * */
function getFlightData(airports, tripType, hasTargets, tripDates) {
  return new Promise(async (resolve) => {
    airports.sort((airport) => {
      if (airport.target !== true) {
        return -1;
      }
    });

    let airportsLength = airports.length;
    let flightDataResult = [];

    for (let i = 0; i < airportsLength; i++) {
      let isTarget = tripType === "RT" ? airports[0]["target"] :
airports[i]["target"];

      if (airports.length == 0 || isTarget === true) {
        break;
      }

      for (let j = 0; j < airports.length; j++) {
        let index = tripType === "RT" ? 0 : i;
```

```

    if (j == index) {
        continue;
    }

    if (airports[j].target !== true && hasTargets === true) {
        continue;
    }

    let departure = tripType === "RT"
        ? airports[0]["IATA"]
        : airports[i]["IATA"];
    let arrival = airports[j]["IATA"];

    const URL = (href =
        "https://el.aegeanair.com/sys/lowfares/routelowfares/?DepartureAirport=" +
        departure +
        "&ArrivalAirport=" +
        arrival +
        "&TripType=" +
        tripType +
        "&DepartureDate=" +
        tripDates.departureDate +
        "&ReturnDate=" +
        tripDates.arrivalDate +
        "&Type=Fares");

    let flights = await getFlights(URL, tripType);

    if (tripType === "RT") {
        addFlights(arrival, departure, flights["Inbound"], tripType,
flightDataResult);
    }
    addFlights(departure, arrival, flights["Outbound"], tripType,
flightDataResult);
}

// Shift the airports array to reduce the number of iterations
if (tripType == "RT") {
    airports.shift();
}
}

if (flightDataResult.length == 0) {
    errorlog.error(`Error Message : Flight data is empty`);
}

```

```

    }
    resolve(flightDataResult);
  });
}

```

Απόσπασμα Κώδικα 4.24: Συνάρτηση getFlightData

Κατά τη συνάρτηση getFlights πραγματοποιούνται τα αιτήματα ή αλλιώς requests προς την ιστοσελίδα της αεροπορικής εταιρείας. Η συνάρτηση έχει ως παραμέτρους εισόδου το διαμορφωμένο URL για κάθε δρομολόγιο καθώς και τον τύπο του ταξιδιού. Στην περίπτωση που ο χρήστης έχει επιλέξει ως τύπο ταξιδιού την επιλογή ταξίδι με επιστροφή (RT), συλλέγονται δεδομένα και για το αντίστροφο δρομολόγιο. Με το τρόπο αυτό, επιτυγχάνεται μείωση του όγκου των αιτημάτων προς την ιστοσελίδα της αεροπορικής εταιρείας κατά 50.6%.

```

/**
 * Returns Outbound and Inbound raw flight data
 *
 * If the trip type option is set to Round Trip (RT), each request collects flight
 * data going both ways (e.g. ATH -> BER, BER -> ATH) to reduce iterations
 * compared to a more bruteforce approach
 *
 * @param {String} url - The endpoint url
 * @param {String} tripType - String that identifies the trip type, either Round
 * Trip (RT) or One Way (OW)
 * @return {Object} flightList - Object containing Outbound and Inbound flight data
 */
async function getFlights(url, tripType) {
  let flightList = { Outbound: [], Inbound: [] };

  try {
    let response = await fetch(url);
    let flights = await response.json();

    flightList["Outbound"] = flights["Outbound"];

    if (tripType === "RT") {
      flightList["Inbound"] = flights["Inbound"];
    }
  } catch (error) {
    errorlog.error(`Error Message : ${error}`);
  } finally {
    return flightList;
  }
}

```

Απόσπασμα Κώδικα 4.25: Συνάρτηση getFlights

Στη συνάρτηση addFlights υπολογίζεται η ημερομηνία πτήσης του κάθε δρομολογίου, η ημερομηνία συλλογής των δεδομένων και τη συνέχεια όλα τα απαραίτητα δεδομένα αποθηκεύονται σε ένα πίνακα με σκοπό την αποθήκευσή τους στη βάση δεδομένων σε μετέπειτα βήμα. Μέσα στο πλήθος δεδομένων που συλλέγονται από την ιστοσελίδα της αεροπορικής εταιρείας, υπάρχει και η ημερομηνία πτήσης του δρομολογίου. Η συγκεκριμένη ημερομηνία παρέχεται με τη μορφή Unix Epoch time (π.χ. 1628208000000). Η μορφή αυτή χρησιμοποιείται συνήθως από λειτουργικά συστήματα και αντιπροσωπεύει τα δευτερόλεπτα που έχουν περάσει από τη 1η Ιανουαρίου 1970 μείον τα άλματα δευτερολέπτων [79]. Για το λόγο αυτό απαιτείται η μετατροπή της σε κανονική μορφή με σκοπό την αποθήκευσή της στη βάση δεδομένων.

```
/**
 * Stores flight data in {Array} flightDataResult ready to be inserted into the
 * database
 *
 * @param {String} departure - The departure airport's IATA
 * @param {String} arrival - The arrival airport's IATA
 * @param {Array<Object>} flights - Array of either Outbound or Inbound flight data
 * @param {String} tripType - String that identifies the trip type, either Round
 * Trip (RT) or One Way (OW)
 * @param {Array} flightDataResult - Array that contains flight data ready to be
 * inserted into the database
 */
function addFlights(departure, arrival, flights, tripType, flightDataResult) {
  if (flights.length !== 0) {
    flights.forEach((flight) => {
      // Convert flightDate into MySQL format
      let flightDate = new
Date(Number(/\d+/.exec(flight["Date"]))).toISOString().slice(0, 19).replace("T", " ");
      let recordDate = new Date().toISOString().slice(0, 19).replace("T", " "); // The
current date

      flightDataResult.push([departure, arrival, flight["Price"], flightDate,
recordDate, tripType]);
    });
  }
}
```

Απόσπασμα Κώδικα 4.26: Συνάρτηση addFlights

Στο τελευταίο στάδιο της διαδικασίας εκτελείται η αποθήκευση των δεδομένων που έχουν συλλεχθεί από τη συνάρτηση `storeData`. Η συνάρτηση αυτή, έχει ως παράμετρο το πίνακα `flightData` που αποτελεί το σύνολο των δεδομένων. Για την αποθήκευση, χρησιμοποιείται ένα ερώτημα SQL με την εντολή `Insert`, ενώ σε περίπτωση αποτυχίας καταγράφεται το αντίστοιχο σφάλμα στο κατάλληλο αρχείο `error.log`.

```
/**
 * Inserts flight data into the database
 *
 * @param {Array<Object>} flightData - Array containing all of the flight info
 */
function storeData(flightData) {
  return new Promise((resolve, reject) => {
    pool.getConnection(function (err, connection) {
      if (err) {
        errorlog.error(`Error Message : ${err}`);
      }

      let sql = "INSERT INTO Flights (departure, arrival, price, flight_date,
record_date, trip_type) VALUES ?";

      connection.query(sql, [flightData], function (error) {
        if (error) {
          errorlog.error(`Error Message : ${error}`);
          reject(error);
        } else {
          connection.release();
          resolve("Task was successful");
        }
      });
    });
  });
}
```

Απόσπασμα Κώδικα 4.27: Συνάρτηση storeData

4.6 Υλοποίηση RESTful API

Η διαδικασία της υλοποίησης του RESTful API βασίστηκε κυρίως στη λειτουργικότητα που προσφέρει το framework Express.js. Το Express προσφέρει κυρίως λειτουργίες για routing καθώς και λειτουργίες για την ανάπτυξη middleware συναρτήσεων.

Αρχικά, πραγματοποιήθηκε η εισαγωγή του Express, δημιουργήθηκε ο εξυπηρετητής, ενώ παράλληλα ορίστηκε η θύρα που θα χρησιμοποιείται για την αλληλεπίδραση με τους εξωτερικούς χρήστες. Σε αυτή την περίπτωση η θύρα παρέχεται από το Heroku και ορίζεται δυναμικά, ως μία μεταβλητή περιβάλλοντος. Σε τοπικό επίπεδο, επιλέγεται η θύρα 3000.

```
const express = require("express");
const { getStatistics } = require("../analytics.js");
const app = express();
const PORT = process.env.PORT || 3000;
```

Απόσπασμα Κώδικα 4.28: Ενσωμάτωση Express.js

Στη συνέχεια δημιουργήθηκαν τα endpoints που θα είναι εκτεθειμένα, όπως αυτά είχαν οριστεί κατά την αρχιτεκτονική σχεδίαση. Πιο συγκεκριμένα κατά το endpoint /results επιστρέφεται ως απόκριση το σύνολο των δεδομένων για όλα τα δρομολόγια. Στην περίπτωση του endpoint /results/{experiment_name} επιστρέφονται μόνο τα δεδομένα για το πείραμα που έχει προσδιοριστεί στη διεύθυνση του HTTP αιτήματος.

Με σκοπό την επεκτασιμότητα της εφαρμογής, δίνεται η δυνατότητα ορισμού των πειραμάτων που έχουν διεξαχθεί, μέσω του πίνακα experiments. Όπως παρουσιάζεται και στη συνέχεια, η μόνη απαίτηση του συστήματος, είναι ο προσδιορισμός του κορμού του κάθε πειράματος, αφού τα δρομολόγια καθώς και τα αποτελέσματα ανακτώνται προγραμματιστικά.

Κατά την έναρξη του εξυπηρετητή, εκτελείται η συνάρτηση handleResults η οποία είναι υπεύθυνη για την προετοιμασία των δεδομένων που θα περιέχουν οι αποκρίσεις. Πιο συγκεκριμένα, καλούνται για κάθε δρομολόγιο οι συναρτήσεις prepareRoutes και prepareStatistics. Η συνάρτηση prepareRoutes είναι υπεύθυνη για τη δημιουργία των δρομολογίων που προκύπτουν από την περιγραφή του πειράματος. Παράλληλα, η συνάρτηση prepareStatistics είναι υπεύθυνη για την ανάκτηση των στατιστικών στοιχείων που έχουν εξαχθεί για κάθε δρομολόγιο καθώς και για την εισαγωγή αυτών στη δομή της απόκρισης. Το κομμάτι της ανάκτησης των στατιστικών στοιχείων παρέχεται από την υπηρεσία ανάλυσης δεδομένων.

```
handleResults();

app.get("/results", async (req, res) => {
  res.send(experiments);
});

app.get("/results/:experiment_name", (req, res) => {
```

```

let experimentName = req.params.experiment_name;
let experiment = getExperiment(experimentName);

res.send(experiment);
});

/**
 * Prepares all of the experiment data
 * that will be sent as a response by the server
 */
function handleResults() {
  experiments.forEach((experiment) => {
    prepareRoutes(experiment);

    prepareStatistics(experiment);
  });
}

```

Απόσπασμα Κώδικα 4.29: Δημιουργία API endpoints

Με αυτό το τρόπο το σύστημα είναι σε θέση να εξυπηρετήσει το ευρύ κοινό και να παρέχει διάφορα στατιστικά στοιχεία καθώς και συμπεράσματα που έχουν εξαχθεί κατά την ανάλυση των δεδομένων. Το σύστημα παραμένει επεκτάσιμο, αφού οι μελλοντικοί ερευνητές και χρήστες της εφαρμογής έχουν τη δυνατότητα να τροποποιήσουν τα προκαθορισμένα endpoints και να παρέχουν με τη σειρά τους, τα δικά τους ευρήματα.

5 Πειραματική Αποτίμηση

Η συλλογή των δεδομένων σχετικά με την τιμή αεροπορικών εισιτηρίων πραγματοποιήθηκε κατά το χρονικό διάστημα 16-31 Αυγούστου για τα πειράματα 1-3 και κατά το χρονικό διάστημα 7-13 Σεπτεμβρίου για το πείραμα 4. Ο συνολικός όγκος δεδομένων που συλλέχθηκαν είναι 9,926 εγγραφές. Αναφορικά με τα πειράματα 1-3, πραγματοποιήθηκε συλλογή σχετικά με την τιμή του εισιτηρίου κάθε ημέρας του Σεπτεμβρίου, σε καθημερινή βάση. Στο πείραμα 4 η διαδικασία αυτή διαφοροποιήθηκε ανάλογα με τις ανάγκες του πειράματος. Πιο συγκεκριμένα, η συλλογή πραγματοποιήθηκε σε πιο τακτά χρονικά διαστήματα, δηλαδή τρεις φορές την ημέρα. Για την παρουσίαση των ευρημάτων του κάθε πειράματος, επιλέχθηκαν ορισμένες περιπτώσεις που παρουσίαζαν ενδιαφέρον, για περαιτέρω εξέταση.

5.1 Ανάλυση Δεδομένων

Ένα από τα πιο σημαντικά μέρη του ερευνητικού σκέλους αυτής της εργασίας, αποτελεί η ανάλυση των δεδομένων. Η διαδικασία αυτή επιτεύχθηκε με την ανάπτυξη των κατάλληλων αλγορίθμων. Πιο συγκεκριμένα, για την εξόρυξη γνώσης από τα δεδομένα επιλέχτηκε η χρήση της περιγραφικής στατιστικής. Τα στατιστικά στοιχεία που αποτέλεσαν τη βάση για τη διαδικασία εξαγωγής συμπερασμάτων, ήταν τα εξής [80]:

Μέση Τιμή

Η μέση αριθμητική τιμή προκύπτει από το πηλίκο του αθροίσματος ενός συνόλου παρατηρήσεων δια του συνολικού πλήθους αυτών. Η μαθηματική σχέση που την εκφράζει είναι η παρακάτω.

$$\bar{x} = \frac{\sum_{i=1}^n x_i}{n}$$

Εξίσωση 5.1: Μέση τιμή

Διάμεσος

Στην περίπτωση που υπάρχουν μία ή περισσότερες, μικρές ή μεγάλες παρατηρήσεις τότε η μέση αριθμητική τιμή δεν είναι αντιπροσωπευτική για τα δεδομένα. Σε μία τέτοια περίπτωση είναι χρήσιμη η διάμεσος. Η διάμεσος αποτελεί την κεντρική παρατήρηση από ένα ταξινομημένο σύνολο τιμών. Αν ο αριθμός των παρατηρήσεων είναι ζυγός αριθμός, τότε η διάμεσος αποτελεί τη μέση τιμή των δύο κεντρικών τιμών.

$$\delta = x_{\left(\frac{n+1}{2}\right)}$$

Εξίσωση 5.2.1: Διάμεσος – Περιττό πλήθος παρατηρήσεων

$$\delta = \frac{x_{\left(\frac{n}{2}\right)} + x_{\left(\frac{n}{2} + 1\right)}}{2}$$

Εξίσωση 5.2.2: Διάμεσος – Ζυγό πλήθος παρατηρήσεων

Διακύμανση

Η διακύμανση είναι ένα από τους τρόπους μέτρησης της διασποράς. Πιο συγκεκριμένα αποτελεί ένα μέγεθος που εκφράζει τη διασπορά των παρατηρήσεων, σε σχέση με τη μέση τιμή τους

$$S^2 = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n}$$

Εξίσωση 5.3: Διακύμανση

Εύρος

Το εύρος αποτελεί και αυτό ένα τα μέτρα διασποράς. Ορίζεται ως η διαφορά της ελάχιστης παρατήρησης από τη μέγιστη και εκφράζεται από την παρακάτω μαθηματική σχέση.

$$R = X_{max} - X_{min}$$

Εξίσωση 5.4: Εύρος

Τυπική Απόκλιση

Η τυπική απόκλιση χρησιμοποιείται και αυτή για τη μέτρηση της διασποράς ή μεταβολής. Το μέτρο αυτό εκφράζει την τυπική απόκλιση των δεδομένων από τη μέση παρατήρηση και ορίζεται ως την τετραγωνική ρίζα της διακύμανσης.

$$s = \sqrt{S^2}$$

Εξίσωση 5.5: Τυπική απόκλιση

Όπως προαναφέρθηκε συλλέχθηκαν δεδομένα για τις ημέρες του Σεπτεμβρίου ανάλογα με τις ανάγκες κάθε πειράματος. Έτσι, για κάθε δρομολόγιο, το πρόγραμμα υπολογίζει τη μέση, την ελάχιστη και τη μέγιστη τιμή του εισιτηρίου για κάθε ημέρα. Στη συνέχεια ακολουθεί μία ντετερμινιστική διαδικασία, κατά την οποία, υπολογίζεται η μέση τιμή, η διάμεσος, η διακύμανση, το εύρος και η τυπική απόκλιση. Με αυτό το τρόπο εξάγονται τα στατιστικά στοιχεία για ολόκληρο το μήνα ή τις μέρες που απαιτεί το αντίστοιχο πείραμα.

Στη συνέχεια, κατά το απόσπασμα κώδικα 5.1, παρουσιάζονται οι συναρτήσεις που χρησιμοποιήθηκαν για τον υπολογισμό των στατιστικών στοιχείων. Πιο συγκεκριμένα, για να υπολογιστεί η μέση τιμή καλείται η συνάρτηση `calcMean` που δέχεται ως όρισμα έναν πίνακα τιμών. Την ίδια παράμετρο δέχεται και η συνάρτηση `calcMedian` που είναι υπεύθυνη για τον υπολογισμό της διαμέσου. Για την εύρεση της διακύμανσης, αναπτύχθηκε η συνάρτηση `calcVariance` που δέχεται ως παραμέτρους, ένα πίνακα τιμών καθώς και τη μέση τιμή. Η συνάρτηση `calcRange` είναι υπεύθυνη για τον υπολογισμό του εύρους τιμών. Τέλος, για την εξεύρεση της τυπικής απόκλισης χρησιμοποιείται η συνάρτηση `calcStandardDeviation`.

```
/**
 * Calculates the mean price
 *
 * @param {Array} prices - Array containing all of the prices of a trip
 * @return {Integer} - Integer representing the mean price
 */
function calcMean(prices) {
  let mean =
    prices.reduce((acc, curr) => {
      return acc + curr;
    }, 0) / prices.length;

  return mean;
}
```

```

/**
 * Calculates the median
 *
 * @param {Array} prices - Array containing all of the prices of a trip
 * @return {Integer} - An integer representing the median price
 */
function calcMedian(prices) {
  const sorted = prices.slice().sort((a, b) => a - b);
  const middle = Math.floor(sorted.length / 2);
  if (sorted.length % 2 === 0) {
    return (sorted[middle - 1] + sorted[middle]) / 2;
  }
  return sorted[middle];
}

/**
 * Calculates the variance of a set of prices
 *
 * @param {Array} prices - Array containing all of the prices of a trip
 * @param {Integer} mean - Integer tha represents the mean value
 * @return {Integer} - Integer representing the variance
 */
function calcVariance(prices, mean) {
  prices = prices.map((k) => {
    return (k - mean) ** 2;
  });

  let sum = prices.reduce((acc, curr) => acc + curr, 0);
  let variance = sum / prices.length - 1;

  return variance;
}

/**
 * Calculates the range of prices
 *
 * @param {Array} prices - Array containing all of the prices of a trip
 * @return {Integer} - Integer that represents the range of prices
 */
function calcRange(prices) {
  prices.sort(function (a, b) {
    return a - b;
  });

```

```

let range = prices[prices.length - 1] - prices[0];
return range;
}

/**
 * Calculates the standard deviation
 *
 * @param {Integer} variance - Integer that represents the variance
 * @return {Integer} - An integer representing the standard deviation
 */
function calcStandardDeviation(variance) {
  return Math.sqrt(variance);
}

```

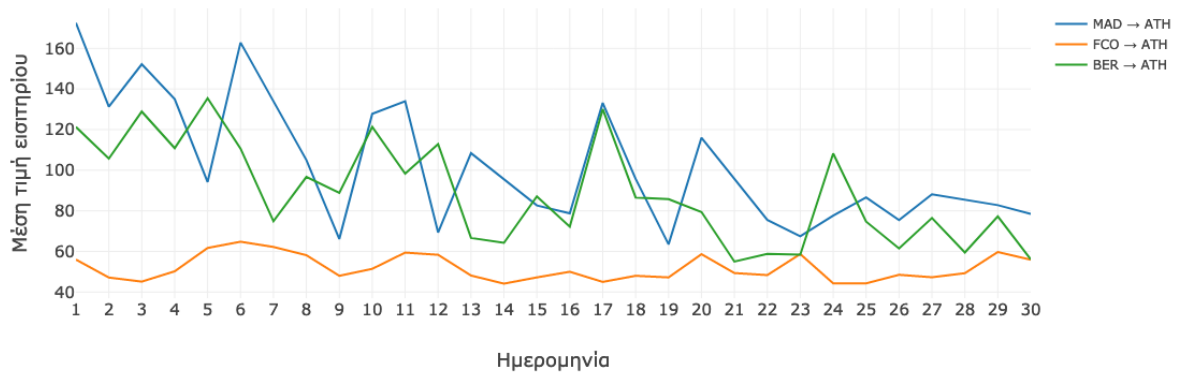
***Απόσπασμα Κώδικα 5.1:** Συναρτήσεις ανάλυσης δεδομένων*

5.2 Πείραμα 1

Ο στόχος αυτού του πειράματος ήταν η παρατήρηση της συμπεριφοράς της τιμής αεροπορικών εισιτηρίων, από πέντε δημοφιλείς ευρωπαϊκές πρωτεύουσες προς την Αθήνα (ATH). Οι πρωτεύουσες που επιλέχθηκαν ήταν η Ρώμη (FCO), το Λονδίνο (LON), η Μαδρίτη (MAD), το Παρίσι (CDG) και το Βερολίνο (BER). Η μελέτη των δεδομένων αυτών, αποσκοπούσε στην εξαγωγή χρήσιμων συμπερασμάτων ή προτύπων.

Αρχικά, κατά την εξέταση των στατιστικών στοιχείων που παρέχει το προτεινόμενο σύστημα, παρατηρήθηκαν πολύ μεγάλες τιμές διακύμανσης στα δρομολόγια με τις υψηλότερες τιμές εισιτηρίων. Πιο συγκεκριμένα, τα πιο ακριβά δρομολόγια από το εξεταζόμενο σύνολο, είναι το Μαδρίτη (MAD) προς Αθήνα (ATH) και το Βερολίνο (BER) προς Αθήνα (ATH). Η διακύμανση των δρομολογίων αυτών είναι 977.59 και 590.28 αντίστοιχα. Σε αντίθεση η διακύμανση των υπόλοιπων δρομολογίων είναι μικρότερη από 60. Παρατηρείται λοιπόν, μία συσχέτιση μεταξύ της μεταβλητότητας και της μέσης τιμής εισιτηρίου. Το σχήμα 5.1 έχει διαμορφωθεί κατάλληλα για την ανάδειξη αυτού του φαινομένου.

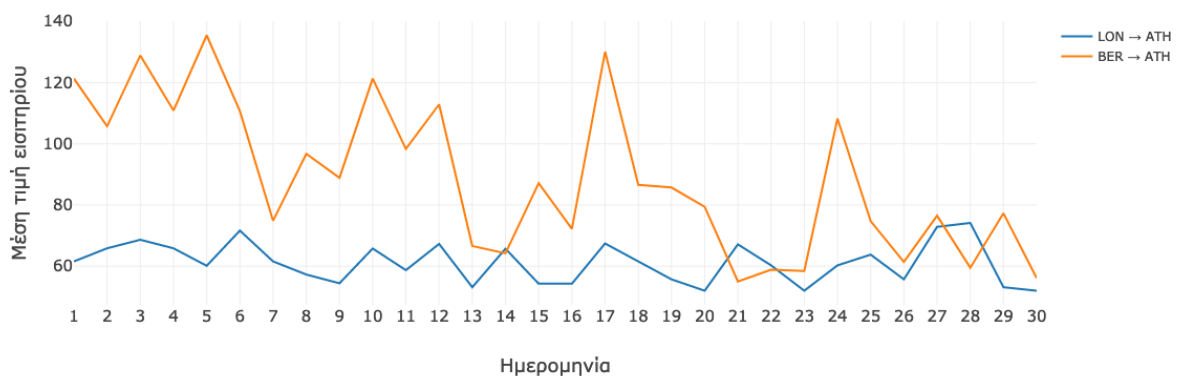
Πείραμα 1 | Μεγαλύτερη διακύμανση τιμής σε ακριβότερα δρομολόγια



Σχήμα 5.1: Πείραμα 1 - Μεγαλύτερη διακύμανση τιμής σε ακριβότερα δρομολόγια

Παράλληλα, κατά τη μελέτη των δεδομένων, παρατηρήθηκε πως για τα δρομολόγια Λονδίνο (LON) προς Αθήνα (ATH) και Βερολίνο (BER) προς Αθήνα (ATH), η μέση τιμή του εισιτηρίου αυξάνεται σε εβδομαδιαία βάση. Πιο συγκεκριμένα, όπως παρουσιάζεται και στο σχήμα 5.2, διακρίνεται αύξηση της τιμής κάθε Παρασκευή, ξεκινώντας από τις 03/09/2021. Οι ημερομηνίες κατά τις οποίες προκύπτει η συγκεκριμένη αύξηση είναι οι 03/09, 10/09, 17/09, 24/09. Τα φαινόμενο αυτό, παρατηρείται μόνο για τα δύο αυτά δρομολόγια και ενδέχεται να οφείλεται στη ζήτηση και στη διαθεσιμότητα των θέσεων της κάθε πτήσης.

Πείραμα 1 | Αύξηση τιμής ανά εβδομάδα

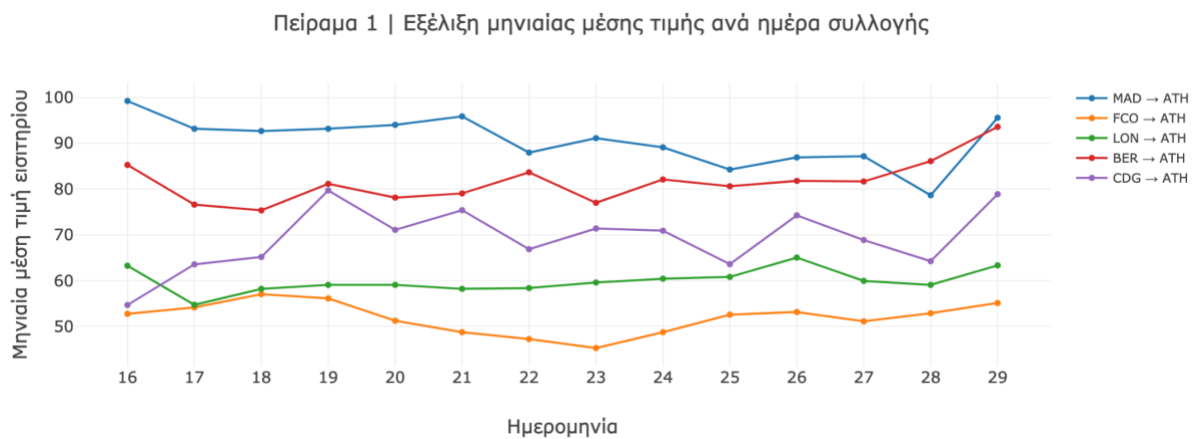


Σχήμα 5.2: Πείραμα 1 - Αύξηση τιμής ανά εβδομάδα

Στη συνέχεια, παρουσιάζονται μέσω του πίνακα 5.1, τα στατιστικά στοιχεία που υπολογίστηκαν κατά το Πείραμα 1, για κάθε δρομολόγιο. Παράλληλα, μέσω του σχήματος 5.3, περιγράφεται η διακύμανση της μηνιαίας μέσης τιμής των εισιτηρίων.

Δρομολόγιο	Μέση Τιμή	Διάμεσος	Διακύμανση	Τυπική Απόκλιση	Εύρος
LON → ATH	61.13	60.85	41.67	6.45	22.14
FCO → ATH	51.88	49.32	36.93	6.07	20.64
MAD → ATH	102.32	91.17	977.59	31.26	109.07
CDG → ATH	69.22	67.75	64.90	8.05	28.57
BER → ATH	88.79	86.14	590.28	24.29	80.42

Πίνακας 5.1: Πείραμα 1 - Στατιστικά στοιχεία δρομολογίων



Σχήμα 5.3: Πείραμα 1 - Διακύμανση μηνιαίας μέσης τιμής

5.3 Πείραμα 2

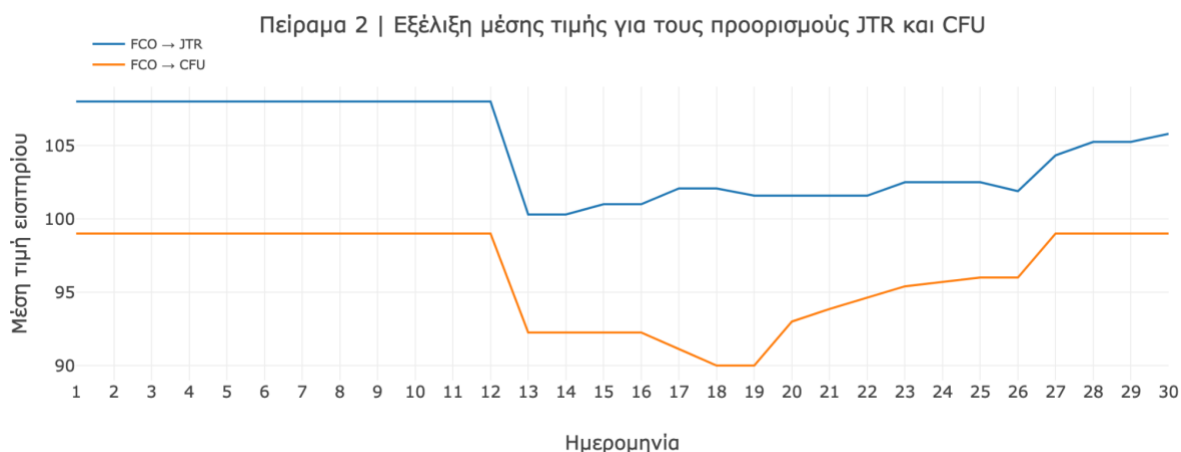
Κατά το Πείραμα 2 κρίθηκε αναγκαία η παρατήρηση της εξέλιξης της τιμής αεροπορικών εισιτηρίων, σχετικά με τις τάσεις του τουρισμού στον ελληνικό χώρο. Για το λόγο αυτό, οι πέντε ευρωπαϊκές πρωτεύουσες από το Πείραμα 1 συνδυάστηκαν με την επιλογή πέντε δημοφιλών ελληνικών νησιών, ώστε να δημιουργηθούν τα κατάλληλα δρομολόγια. Πιο συγκεκριμένα οι προορισμοί που επιλέχθηκαν είναι η Κέρκυρα (CFU), η Ρόδος (RHO), η Σαντορίνη (JTR), η Μήλος (MLO) και η Μύκονος (JMK).

Αναλύοντας τα δεδομένα που συλλέχθηκαν για το συγκεκριμένο πείραμα, διαπιστώθηκε πως το μεγαλύτερο μέρος των δρομολογίων παρουσιάζει μεγάλη διακύμανση στις τιμές των αεροπορικών εισιτηρίων. Ο πίνακας 5.2 περιέχει τα στατιστικά στοιχεία κάποιων ενδεικτικών δρομολογίων.

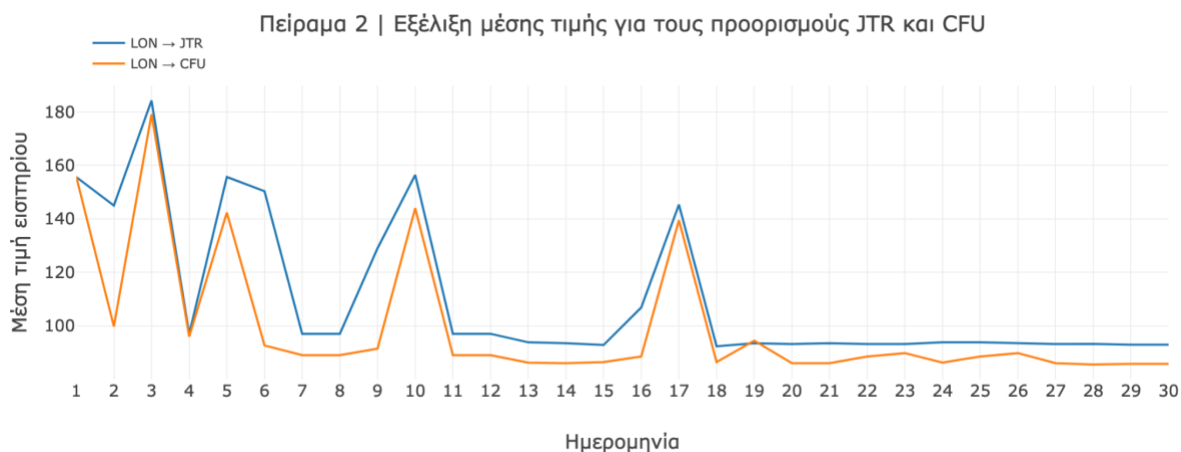
Δρομολόγιο	Μέση Τιμή	Διάμεσος	Διακύμανση	Τυπική Απόκλιση	Εύρος
FCO → JTR	104.63	104.79	8.12	2.84	7.70
FCO → CFU	96.43	99	9.46	3.07	9
LON → JTR	110.19	93.81	720.22	26.83	91.93
LON → CFU	99.41	89	601.46	24.52	93.64
BER → CFU	176.61	101.5	10711.73	103.49	276.63
BER → MLO	221.25	188	5650.81	75.17	266
MAD → JMK	168.11	163.7	858.60	29.30	130.15
CDG → JMK	185.90	157.03	4412.55	66.42	208.07
CDG → RHO	112.45	117	421.20	20.52	77.5

Πίνακας 5.2: Πείραμα 2 - Στατιστικά στοιχεία δρομολογίων

Με την περαιτέρω εξέταση των παραπάνω στατιστικών στοιχείων, παρατηρήθηκε πως η συμπεριφορά της μέσης τιμής για τους προορισμούς Σαντορίνη (JTR) και Κέρκυρα (CFU) ακολουθεί το ίδιο πρότυπο, όταν το σημείο αναχώρησης είναι η Ρώμη (FCO) ή το Λονδίνο (LON). Κατά την εκπόνηση της έρευνας δε βρέθηκε κάποια άλλη συσχέτιση μεταξύ αυτών των προορισμών ή των σημείων αναχώρησης. Συνεπώς, οι δείκτες στους οποίους βασίζεται αυτό το μοτίβο δεν είναι δυνατό να αναγνωριστούν. Στα παρακάτω σχήματα 5.4 και 5.5 παρουσιάζονται οι συσχετίσεις μεταξύ των μέσων τιμών των δρομολογίων.



Σχήμα 5.4: Πείραμα 2 - Εξέλιξη μέσης τιμής για τους προορισμούς Σαντορίνη, Κέρκυρα, με αναχώρηση από τη Ρώμη



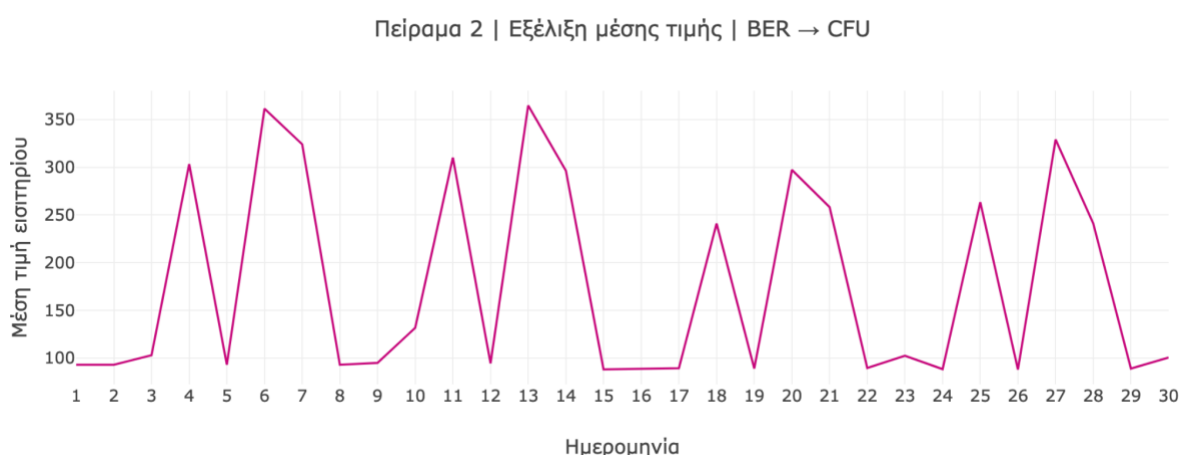
Σχήμα 5.5: Πείραμα 2 - Εξέλιξη μέσης τιμής για τους προορισμούς Σαντορίνη, Κέρκυρα, με αναχώρηση από το Λονδίνο

Μία ακόμη περίπτωση κατά την οποία είναι δυνατή η συσχέτιση δύο δρομολογίων παρουσιάζεται από το σχήμα 5.6. Πιο συγκεκριμένα, τα δρομολόγια με αναχώρηση το Βερολίνο (BER) και προορισμούς τη Μήλο (MLO) και την Κέρκυρα (CFU) φαίνεται να μοιράζονται το ίδιο πρότυπο συμπεριφοράς μέσης τιμής εισιτηρίου. Σε αυτά τα δύο δρομολόγια υπάρχει μεγάλη διακύμανση της μέσης τιμής, ενώ διακρίνονται διαδοχικά διαστήματα αύξησης και μείωσης της. Το μεσοδιάστημα αυτής της συμπεριφοράς, προσεγγίζει κατά μέσο όρο τις τρεις ημέρες.



Σχήμα 5.6: Πείραμα 2 - Εξέλιξη μέσης τιμής για τους προορισμούς Μήλο, Κέρκυρα, με αναχώρηση από το Βερολίνο

Σε αυτό το σημείο, θα εστιάσουμε στο δρομολόγιο Βερολίνo (BER) προς Κέρκυρα (CFU). Το συγκεκριμένο δρομολόγιο είναι άξιο μελέτης αφού παρουσιάζει τη μεγαλύτερη διακύμανση από το εξεταζόμενο σύνολο. Πιο συγκεκριμένα, κατά τις ημέρες 12/09-13/09 διακρίνεται μία αύξηση της μέσης τιμής κατά 287%. Η αύξηση αυτή, αποτελεί τη μεγαλύτερη ποσοστιαία αύξηση που παρατηρήθηκε κατά την εκπόνηση της παρούσας μελέτης. Ένα ακόμα ενδιαφέρον στοιχείο, αποτελεί το μοτίβο που ακολουθεί η μέση τιμή. Σε αυτή την περίπτωση γίνεται ακόμη πιο ξεκάθαρη η εβδομαδιαία συμπεριφορά αύξησης και μείωσης της. Συμπεραίνεται, πως για κάθε εβδομάδα του μήνα, η μέση τιμή αυξάνεται τις ημέρες Σάββατο και Δευτέρα ενώ μειώνεται την ημέρα της Κυριακής. Όπως εμφανίζεται στο σχήμα 5.7, η πρώτη παρατήρηση αυτής της συμπεριφοράς ξεκινά την Παρασκευή 04/09. Η ύπαρξη ενός τέτοιου προτύπου, ενδέχεται να οφείλεται στις προτιμήσεις των ταξιδιωτών, δηλαδή, στη μεγαλύτερη ζήτηση εισιτηρίων κατά το τέλος της εβδομάδας.



Σχήμα 5.7: Πείραμα 2 - Εξέλιξη μέσης τιμής για το δρομολόγιο Βερολίνo προς Κέρκυρα

5.4 Πείραμα 3

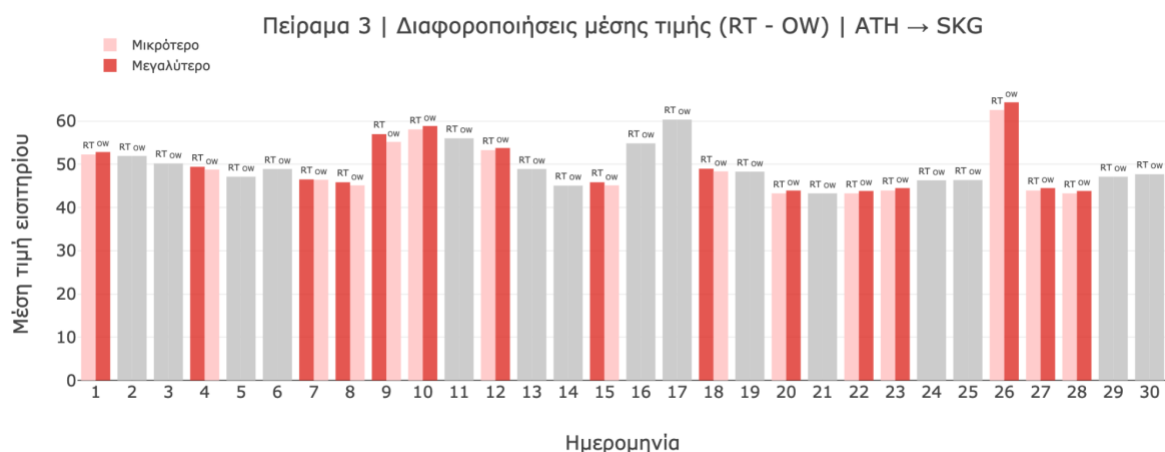
Ο στόχος του Πειράματος 3 αποτέλεσε η εξέταση της πορείας της τιμής ανάμεσα σε δρομολόγια με επιστροφή και χωρίς επιστροφή (μονή διαδρομή). Πιο συγκεκριμένα, αποφασίστηκε πως θα ήταν ενδιαφέρουσα η εξαγωγή συμπερασμάτων σχετικά με τυχόν διαφοροποιήσεις που θα μπορούσαν να προκύψουν σε τέτοιου είδους ταξίδια. Για το λόγο αυτό, συλλέχθηκαν δεδομένα για το δρομολόγιο Αθήνα (ATH) προς Θεσσαλονίκη (SKG), και το αντίστροφο. Η συλλογή αυτή περιλαμβάνει δρομολόγια με επιστροφή (RT) και χωρίς επιστροφή (OW).

Σε αυτό το πείραμα, όπως εκφράζεται από τον πίνακα 5.2, υπάρχουν πολύ μικρές αποκλίσεις μεταξύ των εξεταζόμενων τύπων ταξιδιών. Πιο συγκεκριμένα, στην περίπτωση του δρομολογίου Αθήνα (ATH) προς Θεσσαλονίκη (SKG), η επιλογή του τύπου ταξιδιού χωρίς επιστροφή (OW) είναι πιο ακριβή από την επιλογή με επιστροφή (RT). Σε αντίθεση, στο δρομολόγιο Θεσσαλονίκη (SKG) προς Αθήνα (ATH) η επιλογή του τύπου ταξιδιού με επιστροφή (RT) είναι πιο ακριβή από την επιλογή του τύπου χωρίς επιστροφή (OW). Παρατηρείται επίσης, πως η διάμεσος και στις δύο περιπτώσεις δρομολογίων είναι η ίδια. Το γεγονός αυτό μας οδηγεί στο συμπέρασμα πως το σύνολο των δεδομένων ακολουθούν κανονική κατανομή, χωρίς ακραίες τιμές.

Δρομολόγιο	Μέση Τιμή	Διάμεσος	Διακύμανση	Τυπική Απόκλιση	Εύρος
ATH → SKG (RT)	49.33	48	26.52	5.15	19.28
ATH → SKG (OW)	49.40	48	27.47	5.24	21.07
SKG → ATH (RT)	40.09	38.92	82.44	9.08	36.57
SKG → ATH (OW)	39.89	38.92	81.63	9.03	36.71

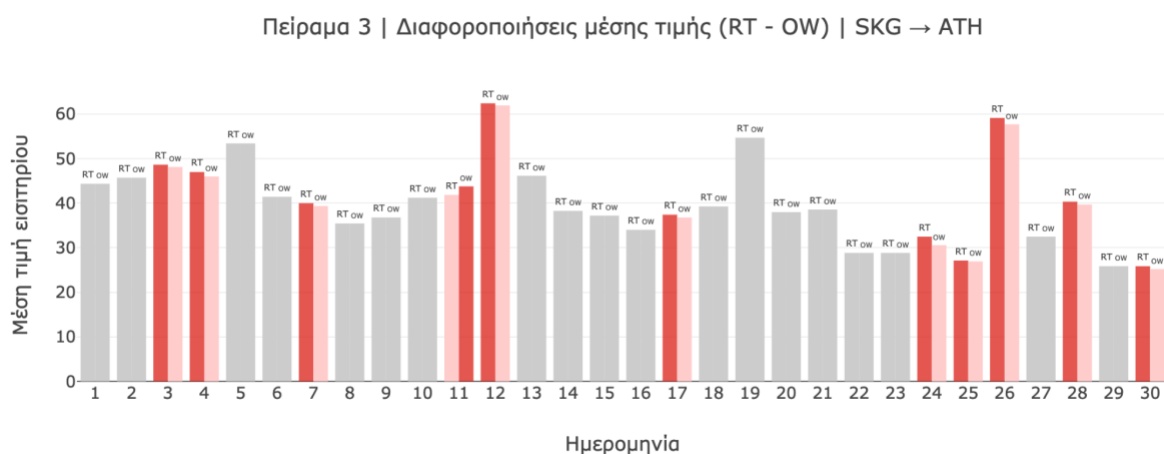
Πίνακας 5.3: Πείραμα 3 - Στατιστικά στοιχεία δρομολογίων

Στις σχηματικές απεικονίσεις 5.8 και 5.9 παρουσιάζονται οι διαφοροποιήσεις των μέσων τιμών των εισιτηρίων, μεταξύ δρομολογίων με επιστροφή και χωρίς επιστροφή. Στην περίπτωση του δρομολογίου Αθήνα (ATH) προς Θεσσαλονίκη (SKG), κατά το σχήμα 5.8, ο τύπος ταξιδιού με επιστροφή είναι πιο ακριβός, έξι ημέρες τον μήνα, ενώ ο τύπος, χωρίς επιστροφή εννιά ημέρες. Κατά το μεγαλύτερο διάστημα του μήνα, οι τιμές είναι ίσες. Η μεγαλύτερη διαφορά παρατηρήθηκε στις 09/09 και ήταν της τάξης των 2 ευρώ.



Σχήμα 5.8: Πείραμα 3 - Διαφοροποιήσεις μέσης τιμής, Αθήνα προς Θεσσαλονίκη (RT - OW)

Το σχήμα 5.9 αναφέρεται στο δρομολόγιο Θεσσαλονίκη (SKG) προς Αθήνα (ATH). Σε αυτή την περίπτωση, παρατηρούνται ακόμη λιγότερες διαφοροποιήσεις ανάμεσα στους τύπους των δρομολογίων. Πιο συγκεκριμένα, τις περισσότερες ημέρες του μήνα οι τιμές παραμένουν ίσες, τα δρομολόγια με επιστροφή είναι πιο ακριβά για δέκα ημέρες, ενώ τα μονής διαδρομής είναι πιο ακριβά για μόλις μία ημέρα. Η μεγαλύτερη διαφορά που παρατηρήθηκε ήταν και σε αυτή την περίπτωση της τάξεως των 2 ευρώ. Η διαφορά αυτή προκύπτει στις 24/09.



Σχήμα 5.9: Πείραμα 3 - Διαφοροποιήσεις μέσης τιμής, Θεσσαλονίκη προς Αθήνα (RT - OW)

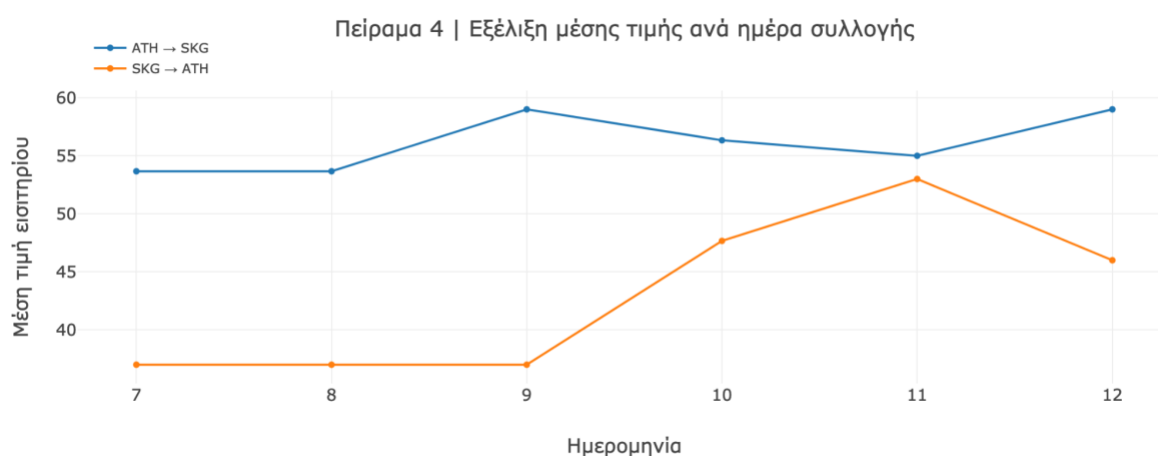
Συμπεραίνεται λοιπόν, πως η διαφορά μεταξύ δρομολογίων με επιστροφή και χωρίς επιστροφή είναι αμελητέα. Από το σύνολο των συλλεχθέντων δεδομένων δεν μπορεί να εξαχθεί κάποιο πρότυπο για τη συμπεριφορά της τιμής ανάμεσα σε αυτούς τους δύο τύπους ταξιδιών.

5.5 Πείραμα 4

Τέλος, το πείραμα 4 αποτέλεσε μία προσπάθεια ανακάλυψης γνώσης σχετικά με την εξέλιξη της τιμής ενός εισιτηρίου σε επίπεδο ημέρας, μέχρι την αναχώρηση του δρομολογίου. Για τον σκοπό αυτό επιλέχθηκαν δύο δρομολόγια, από Αθήνα (ATH) προς Θεσσαλονίκη (SKG), και το αντίστροφο. Η ημέρα αναχώρησης των δρομολογίων αυτών ήταν η 13 Σεπτεμβρίου. Έτσι, συλλέχθηκαν δεδομένα τρεις φορές την ημέρα σε ένα χρονικό διάστημα έξι ημερών, καθώς η ημέρα έναρξης ήταν η 7 Σεπτεμβρίου.

Στο πρώτο δρομολόγιο Αθήνα (ATH) προς Θεσσαλονίκη (SKG) διακρίθηκε μία συχνή διακύμανση της τιμής των εισιτηρίων, χωρίς όμως αυτή να αντιστοιχεί σε μεγάλη ποσοστιαία αύξηση ή μείωση της. Πιο συγκεκριμένα, η τιμή διαφοροποιήθηκε συνολικά εννιά φορές, με μέσο όρο 1.5 αλλαγές την ημέρα. Το ποσοστό κατά το οποίο η τιμή διακυμαινόταν, ήταν 15.6%.

Κατά τη δεύτερη περίπτωση, δηλαδή του δρομολογίου Θεσσαλονίκη (SKG) προς Αθήνα (ATH) παρατηρήθηκε πολύ μικρή συχνότητα διακύμανσης, ενώ παράλληλα υπήρχε μεγάλο ποσοστό διακύμανσης. Ειδικότερα, η τιμή διακυμάνθηκε συνολικά δύο φορές σε έξι ημέρες συλλογής δεδομένων. Σε αντίθεση με το πρώτο δρομολόγιο, η τιμή αυξήθηκε μία φορά κατά 43.2%, και μειώθηκε κατά 13.2%. Έτσι λοιπόν, συμπεραίνουμε πως το δεύτερο δρομολόγιο είχε μεγαλύτερο ποσοστό διακύμανσης. Λόγω, των διαφοροποιήσεων ανάμεσα στα δύο αυτά δρομολόγια δεν προκύπτει η ύπαρξη κάποιου γενικευμένου προτύπου σχετικά με τη συμπεριφορά της τιμής σε ημερήσια βάση.



Σχήμα 5.10: Πείραμα 4 - Εξέλιξη μέσης τιμής ανά ημέρα συλλογής

Δρομολόγιο	Μέση Τιμή	Διάμεσος	Διακύμανση	Τυπική Απόκλιση	Εύρος
ATH → SKG	56.11	55.66	3.98	1.99	5.33
SKG → ATH	43.71	46	36.66	6.05	16

Πίνακας 5.4: Πείραμα 4 - Στατιστικά στοιχεία δρομολογίων

6 Επίλογος

6.1 Σύνοψη και Συμπεράσματα

Συνοψίζοντας, η παρούσα πτυχιακή εργασία εστίασε στην διαδικασία εξόρυξης γνώσης από δεδομένα μεγάλου όγκου στο χώρο των Μέσων Μαζικής Μεταφοράς και ειδικότερα σε αυτόν των αερομεταφορών. Για το λόγο αυτό πραγματοποιήθηκε η υλοποίηση ενός συστήματος που έχει τη δυνατότητα να συλλέγει δεδομένα, να τα αναλύει και να διαθέτει τα ευρήματα της ανάλυσης αυτής στο ευρύ κοινό, μέσω ενός Web API. Κατά τη διαδικασία ανάπτυξης του προτεινόμενου συστήματος χρησιμοποιήθηκαν τεχνικές και τεχνολογίες που επιτρέπουν μεγάλη επεκτασιμότητα αλλά και ευχρηστία. Με αυτό το τρόπο, το προτεινόμενο σύστημα μπορεί πολύ εύκολα να τροποποιηθεί κατάλληλα, ώστε να ανταποκρίνεται στις ανάγκες της εκάστοτε έρευνας. Αναφορικά με το πειραματικό σκέλος της εργασίας, έγινε προσπάθεια εξόρυξης χρήσιμης πληροφορίας από δεδομένα που αφορούν την τιμή εισιτηρίων, αεροπορικών δρομολογίων που έχουν ως προορισμούς ελληνικές πόλεις. Ειδικότερα, επιλέχθηκαν τέσσερις ενδιαφέρουσες περιπτώσεις, κατά τις οποίες μελετήθηκε η συμπεριφορά της τιμής και στη συνέχεια εξάχθηκαν συμπεράσματα και πρότυπα για αυτή.

Σύμφωνα με τα ευρήματα της παρούσας έρευνας, μπορεί να συμπεράνει κανείς, πως η ύπαρξη ενός επιμελητή – παρατηρητή κατά τη διαδικασία εξόρυξης χρήσιμης πληροφορίας από δεδομένα μεγάλου όγκου, είναι ιδιαίτερα σημαντική. Σε αντίθεση με ένα εξ' ολοκλήρου αυτοματοποιημένο σύστημα, η φύση του ανθρώπινου παράγοντα, επιτρέπει την άμεση εξαγωγή συμπερασμάτων και προτύπων. Μέσα σε ένα τέτοιο πλαίσιο λοιπόν, διαπιστώνεται πως η εξόρυξη γνώσης από μεγάλα σύνολα δεδομένων είναι εφικτή αλλά και άκρως απαραίτητη για την αποτελεσματικότερη λήψη αποφάσεων σε όλους τους τομείς. Αναφορικά με το χώρο των αερομεταφορών, οι ταξιδιώτες είναι σε θέση να λάβουν αποτελεσματικότερες αποφάσεις σχετικά με την αγορά εισιτηρίων αλλά και να προγραμματίσουν πιο οικονομικά τα ταξίδια τους.

6.2 Μελλοντικές Επεκτάσεις

Η αδιάλειπτη διαδικασία βελτίωσης αποτελεί ένα από τα πιο σημαντικά χαρακτηριστικά ενός συστήματος ή μιας έρευνας. Έτσι λοιπόν, το σύστημα που αναπτύχθηκε στα πλαίσια αυτής της πτυχιακής εργασίας διαθέτει περιθώρια βελτίωσης. Πιο συγκεκριμένα, ως μελλοντική επέκταση, θα μπορούσε να αναπτυχθεί μία δυναμική ιστοσελίδα, μέσω της οποίας να παρουσιάζονται τα γραφήματα των αποτελεσμάτων καθώς και τα συμπεράσματα που έχουν εξαχθεί. Στο ίδιο πλαίσιο ο χρήστης θα μπορούσε να κάνει λήψη των προαναφερθέντων ως μία σύντομη αναφορά σε μορφή PDF. Παράλληλα θα ήταν χρήσιμη η επέκταση του συστήματος για να υποστηρίζει τη συλλογή αλλά και την ανάλυση δεδομένων σε πραγματικό χρόνο.

Κατά το ερευνητικό σκέλος της εργασίας, θα ήταν ιδιαίτερα ενδιαφέρον να επεκταθεί το χρονικό διάστημα συλλογής των δεδομένων. Με το τρόπο αυτό, θα ήμασταν σε θέση να εξάγουμε πιο εμπειριστατωμένα αποτελέσματα, αφού θα υπήρχαν περισσότερα σημεία αναφοράς. Παράλληλα, θα ήταν αρκετά χρήσιμη η συλλογή δεδομένων για περισσότερους προορισμούς καθώς θα μας επέτρεπε να δημιουργήσουμε μία πιο σφαιρική εικόνα των δεδομένων. Τέλος, το ερευνητικό μέρος θα μπορούσε να συνδυαστεί με την ανάπτυξη ενός συστήματος πρόβλεψης τιμών αεροπορικών εισιτηρίων.

Βιβλιογραφία

1. W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus, “Knowledge Discovery in Databases: An Overview”, *AIMag*, vol. 13, no. 3, p. 57, Sep. 1992, doi: 10.1609/aimag.v13i3.1011.
2. U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, “From Data Mining to Knowledge Discovery in Databases”, *AIMag*, vol. 17, no. 3, p. 37, Mar. 1996, doi: 10.1609/aimag.v17i3.1230
3. F. Branda, F. Marozzo, and D. Talia, “Ticket Sales Prediction and Dynamic Pricing Strategies in Public Transport,” *Big Data and Cognitive Computing*, vol. 4, no. 4, p. 36, Nov. 2020, doi: 10.3390/bdcc4040036.
4. O. Etzioni, R. Tuchinda, C. A. Knoblock, and A. Yates, “To buy or not to buy,” *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '03*, 2003, doi: 10.1145/956750.956767.
5. W. Groves and M. Gini, “On Optimizing Airline Ticket Purchase Timing,” *ACM Transactions on Intelligent Systems and Technology*, vol. 7, no. 1, pp. 1–28, Oct. 2015, doi: 10.1145/2733384.
6. T. Janseen, “A Linear Quantile Mixed Regression Model for Prediction of Airline Ticket Prices,” Radboud University, 2014.
7. Aegean Airlines, “Προφίλ | About Aegean,” Aegean Airlines.
<https://el.about.aegeanair.com/etaireia/profil/>
8. F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč, “Foundations of JSON Schema,” *Proceedings of the 25th International Conference on World Wide Web*, Apr. 2016, doi: 10.1145/2872427.2883029.

9. The Linux Foundation, "Database Definition," [linfo.org](http://linfo.org/database.html), Jun. 22, 2006.
<http://linfo.org/database.html>.
10. R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 7th ed. Pearson, 2016.
11. M. Madison, M. Barnhill, C. Napier, and J. Godin, "NoSQL Database Technologies," *Journal of International Technology and Information Management Journal of International Techn*, vol. 24, no. 1, 2015, Accessed: Aug. 09, 2021. [Online]. Available: <https://scholarworks.lib.csusb.edu/jitim/vol24/iss1/1>.
12. A. B. M. Moniruzzaman and S. A. Hossain, "NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison," *International Journal of Database Theory and Application*, vol. 6, no. 4, 2013, Accessed: Aug. 09, 2021. [Online]. Available: <https://arxiv.org/ftp/arxiv/papers/1307/1307.0191.pdf>.
13. MySQL, "MySQL Customers by Industry" www.mysql.com.
<https://www.mysql.com/customers/industry/?id=> (accessed Aug. 10, 2021).
14. S. Pachev, *Understanding MySQL internals*. Sebastopol, Calif.: O'reilly, 2007.
15. C. Bell, *Expert MySQL*. Berkley: Apress, 2012.
16. Amazon, "Amazon Relational Database Service (RDS)," *Amazon Web Services, Inc.*, 2019. <https://aws.amazon.com/rds/> (accessed Aug. 10, 2021).
17. Azure, "Azure Database for MySQL," *Microsoft.com*, 2019.
<https://azure.microsoft.com/en-us/services/mysql/> (accessed Aug. 10, 2021).
18. Heroku, "ClearDB MySQL | Heroku Dev Center," *devcenter.heroku.com*.
<https://devcenter.heroku.com/articles/cleardb> (accessed Aug. 10, 2021)
19. Mozilla, "What is JavaScript?," *MDN Web Docs*, Jul. 02, 2019.
https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript (accessed Aug. 11, 2021).

20. Ecma International, “ECMAScript® 2015 Language Specification,” ECMA-262 6th Edition, 2015, Accessed: Aug. 11, 2021. [Online]. Available: https://www.ecma-international.org/wp-content/uploads/ECMA-262_6th_edition_june_2015.pdf.
21. C. Severance, “JavaScript: Designing a Language in 10 Days,” *Computer*, vol. 45, no. 2, pp. 7–8, Feb. 2012, doi: 10.1109/MC.2012.57.
22. B. Eich, “JavaScript creator ponders past, future,” Jun. 23, 2008.
23. Stack Overflow, “Stack Overflow Developer Survey 2021,” Stack Overflow, 2021. <https://insights.stackoverflow.com/survey/2021#technology-most-popular-technologies> (accessed Aug. 12, 2021).
24. S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to Build High-Performance Network Programs," in *IEEE Internet Computing*, vol. 14, no. 6, pp. 80-83, Nov.-Dec. 2010, doi: 10.1109/MIC.2010.145.
25. Mozilla, “Promise,” MDN Web Docs, Mar. 18, 2019. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise (accessed Aug. 14, 2021).
26. M. Madsen, O. Lhoták, and F. Tip, “A model for reasoning about JavaScript promises,” *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, pp. 1–24, Oct. 2017, doi: 10.1145/3133910.
27. Pyrolisticalin, “How to escape Promise Hell,” Steemit, Jul. 27, 2016. <https://steemit.com/javascript/@pyrolistical/how-to-escape-promise-hell> (accessed Aug. 14, 2021).
28. Ecma International, “ECMAScript® 2017 Language Specification,” ECMA-262 8th Edition, 2017, Accessed: Aug. 14, 2021. [Online]. Available: <https://262.ecma-international.org/8.0/>.

29. R. Gandhi, “Seemingly Imperative with async and await,” JavaScript Next, pp. 211–222, 2019, doi: 10.1007/978-1-4842-5394-6_14.
30. Ilya Kantor, “Async/await,” Javascript.info, 2020. <https://javascript.info/async-await> (accessed Aug. 15, 2021).
31. Node.js Foundation, “Overview of Blocking vs Non-Blocking | Node.js,” Node.js, 2019. <https://nodejs.org/en/docs/guides/blocking-vs-non-blocking/> (accessed Aug. 16, 2021).
32. rromoff, “From streaming to studio: The evolution of Node.js at Netflix,” OpenJS Foundation, Sep. 24, 2020. <https://openjsf.org/blog/2020/09/24/from-streaming-to-studio-the-evolution-of-node-js-at-netflix/> (accessed Aug. 16, 2021).
33. L. Lozinski, “The Uber Engineering Tech Stack, Part II: The Edge and Beyond,” Uber Engineering Blog, Jul. 21, 2016. <https://eng.uber.com/uber-tech-stack-part-two/> (accessed Aug. 11, 2021).
34. libuv, “libuv/Documentation,” GitHub, Oct. 19, 2019. <https://github.com/libuv/libuv> (accessed Aug. 17, 2019).
35. “Documentation · V8,” v8.dev. <https://v8.dev/docs> (accessed Aug. 17, 2021).
36. npmjs, “package.json | npm Docs,” docs.npmjs.com. <https://docs.npmjs.com/cli/v7/configuring-npm/package-json> (accessed Aug. 17, 2021).
37. npmjs, “Downloading and installing packages locally | npm Docs,” docs.npmjs.com. <https://docs.npmjs.com/downloading-and-installing-packages-locally> (accessed Aug. 17, 2021).
38. npmjs, “npm Blog Archive: So long, and thanks for all the packages!,” blog.npmjs.org. <https://blog.npmjs.org/post/615388323067854848/so-long-and-thanks-for-all-the-packages.html> (accessed Aug. 17, 2021).

39. Node.js, “The Node.js Event Loop, Timers, and process.nextTick(),” Node.js. <https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/> (accessed Aug. 18, 2021).
40. H. Sun, D. Bonetta, F. Schiavio, and W. Binder, “Reasoning about the Node.js Event Loop using Async Graphs,” IEEE Xplore, Feb. 01, 2019. <https://ieeexplore.ieee.org/document/8661173> (accessed Aug. 18, 2021).
41. D. H. Curie, J. Jaison, J. Yadav, and J. R. Fiona, “Analysis on Web Frameworks,” Journal of Physics: Conference Series, vol. 1362, p. 012114, Nov. 2019, doi: 10.1088/1742-6596/1362/1/012114.
42. Express.js, “Companies using Express,” [expressjs.com](https://expressjs.com/en/resources/companies-using-express.html). <https://expressjs.com/en/resources/companies-using-express.html> (accessed Aug. 19, 2021).
43. C. Peters, “Building Rich Internet Applications with Node.js and Express.js,” Carl von Ossietzky University of Oldenburg.
44. Express.js, “Express basic routing,” [expressjs.com](https://expressjs.com/en/starter/basic-routing.html). <https://expressjs.com/en/starter/basic-routing.html> (accessed Aug. 19, 2021).
45. N. Middleton and R. Schneeman, Heroku: Up and Running: Effortless Application Deployment and Scaling. Beijing ; Köln: O’reilly, 2014.
46. P. M. Mell and T. Grance, “The NIST definition of cloud computing” 2011, doi: 10.6028/nist.sp.800-145.
47. Anubhav Hanjura, Heroku Cloud Application Development. Packt Publishing Limited, 2014.

48. Heroku, “Heroku Development Languages: Programming Languages Heroku supports | Heroku,” Heroku.com, 2020. <https://www.heroku.com/languages> (accessed Aug. 20, 2021).
49. Heroku, “Heroku Dynos | Heroku,” Heroku.com, 2019. <https://www.heroku.com/dynos> (accessed Aug. 20, 2021).
50. Heroku, “Dynos and the Dyno Manager | Heroku Dev Center,” devcenter.heroku.com. <https://devcenter.heroku.com/articles/dynos> (accessed Aug. 20, 2021).
51. J. Ansaharju, “Improving Software Development with Platform-as-a-Service Product – Using Heroku in Web Application Project,” MSc Thesis, Helsinki Metropolia University of Applied Sciences, 2016.
52. Heroku, “The Process Model | Heroku Dev Center,” devcenter.heroku.com. <https://devcenter.heroku.com/articles/process-model> (accessed Sep. 10, 2021).
53. Heroku, “How Heroku Works | Heroku Dev Center,” devcenter.heroku.com. <https://devcenter.heroku.com/articles/how-heroku-works> (accessed Aug. 20, 2021).
54. IBM, “DBaaS (Database-as-a-Service),” www.ibm.com. <https://www.ibm.com/cloud/learn/dbaas> (accessed Aug. 21, 2021).
55. NAVISITE, “ClearDB Developer Center - Welcome,” www.cleardb.com. <https://www.cleardb.com/developers/platform/overview> (accessed Aug. 21, 2021).
56. R. Fielding et al., “RFC 2616, Hypertext Transfer Protocol -- HTTP/1.1.” Jun. 1999, doi: 10.17487/rfc2616.
57. T. Berners-Lee, R. Fielding, and L. Masinter, “Uniform Resource Identifiers (URI): Generic Syntax,” Aug. 1998, doi: 10.17487/rfc2396.
58. A. S. Tanenbaum and D. Wetherall, Computer Networks, Fifth Edition. Prentice Hall, 2010.

59. L. Dusseault and J. Snell, “PATCH Method for HTTP,” Mar. 2010, doi: 10.17487/rfc5789.
60. Mozilla, “An overview of HTTP,” MDN Web Docs, Feb. 19, 2019. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview> (accessed Sep. 09, 2021).
61. IBM, “What is a REST API?,” www.ibm.com. <https://www.ibm.com/cloud/learn/rest-apis> (accessed Aug. 24, 2021).
62. M. Biehl, RESTful API design : APIs your consumers will love. Api-University Press, 2016.
63. R. T. Fielding, “REST: Architectural Styles and the Design of Network-based Software Architectures,” PhD dissertation, University of California, Irvine, 2000.
64. F. Doglio, Rest API development with Node.JS : manage and understand the full capabilities of successful REST development. New York, Ny: Apress, 2018.
65. International Air Transport Association (IATA), “IATA Airline and Location Codes,” www.iata.org. <https://www.iata.org/en/services/codes/> (accessed Aug. 27, 2021).
66. International Organization for Standardization (ISO), “ISO 3166 — Country Codes,” ISO. <https://www.iso.org/iso-3166-country-codes.html> (accessed Aug. 27, 2021).
67. L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Upper Saddle River (N.J.): Addison-Wesley, 2013.
68. A. Wiggins, “The Twelve-Factor App,” 12factor.net, 2017. <https://12factor.net/config> (accessed Aug. 28, 2021).

69. Node.js, “Node.js v16.9.0 Documentation,” [nodejs.org](https://nodejs.org/docs/latest/api/process.html#process_process_env).
https://nodejs.org/docs/latest/api/process.html#process_process_env (accessed Aug. 29, 2021).
70. Node Fetch Team, “node-fetch: A light-weight module that brings the Fetch API to Node.js,” GitHub. <https://github.com/node-fetch/node-fetch> (accessed Aug. 29, 2021).
71. Heroku, “Cloud Application Platform | Heroku,” *Heroku.com*, May 03, 2019. <https://www.heroku.com/home> (accessed Aug. 29, 2021).
72. Ravishankar Somasundaram, *Git: version control for everyone beginner’s guide*. Birmingham: Packt Publ, 2013.
73. mysql, “MySQL,” *Mysql.com*, 2000. <https://www.mysql.com/> (accessed Aug. 30, 2021).
74. Node Cron, “node-cron,” npm. <https://www.npmjs.com/package/node-cron> (accessed Aug. 29, 2021).
75. mysql, “mysql,” npm. <https://www.npmjs.com/package/mysql> (accessed Aug. 30, 2021).
76. Joyent, “Error Handling in Node.js,” *Joyent*. <https://www.joyent.com/node-js/production/design/errors> (accessed Aug. 31, 2021).
77. Mozilla, “Control flow and error handling,” [developer.mozilla.org](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling).
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling (accessed Aug. 30 2021).
78. Mozilla, “try...catch,” MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch> (accessed Sep. 02, 2021).
79. K. Thompson and D. Ritchie, *UNIX PROGRAMMER’S MANUAL*, 4th ed. Bell Telephone Laboratories, Inc., 1973.

80. G. Vlachopoulos, G. Lagoumintzis, and K. Koutsogiannis, “Περιγραφική Στατιστική,” *Μεθοδολογία της έρευνας στις επιστήμες υγείας*, vol. 6, Mar. 2016, doi: <http://hdl.handle.net/11419/5362>.