ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΡΟΒΛΗΜΑ ΔΡΟΜΟΛΟΓΗΣΗΣ ΟΧΗΜΑΤΩΝ: ΜΙΑ WEB ΕΦΑΡΜΟΓΗ ΕΠΙΛΥΣΗΣ

Διπλωματική Εργασία

του

Σταματίου Αγησιλάου-Θεοδώρου

Θεσσαλονίκη, Φεβρουάριος 2019

ΠΡΟΒΛΗΜΑ ΔΡΟΜΟΛΟΓΗΣΗΣ ΟΧΗΜΑΤΩΝ: ΜΙΑ WEB ΕΦΑΡΜΟΓΗ ΕΠΙΛΥΣΗΣ

Σταματίου Αγησίλαος-Θεόδωρος

Πτυχίο Εφαρμοσμένης Πληροφορικής – Απρίλιος 2016

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής
Χατζηγεωργίου Αλέξανδρος

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 27/02/2019

Χατζηγεωργίου Αλέξανδρος	Γεωργιάδης Χρήστος	Στειακάκης Εμμανουήλ
.....	.....	.....

Σταματίου Αγησίλαος-Θεόδωρος

.....

Περίληψη

Σκοπός της εργασίας είναι η ανάπτυξη web εφαρμογής για την επίλυση του προβλήματος της δρομολόγησης οχημάτων, στο πλαίσιο της υποστήριξης των διαδικασιών της εφοδιαστικής αλυσίδας, με την τεχνολογία Java EE. Εκπαιδευτικός στόχος είναι η εξοικείωση και η διερεύνηση των τεχνολογιών που παρέχονται από την έκδοση της Java EE και η μελέτη των σύγχρονων τάσεων στην ανάπτυξη διαδικτυακών εφαρμογών.

Λαμβάνοντας υπόψη ένα σύνολο οχημάτων, ένα σύνολο σταθμών ανεφοδιασμού και ένα σύνολο θέσεων που χρειάζονται εξυπηρέτηση (Pick up/Delivery) καθώς και της απόστασης των οχημάτων και των θέσεων, θα αναπτυχθεί μια εύχρηστη web εφαρμογή από την οποία θα γίνεται η κατανομή θέσεων σε οχήματα ανά παραγγελία ώστε να ελαχιστοποιείται η συνολική απόσταση του οχήματος που πρέπει να διανυθεί. Για την επίλυση του προβλήματος θα γίνει χρήση έτοιμων βιβλιοθηκών ανοιχτού κώδικα σε Java προσανατολισμένων στην λύση αντίστοιχων προβλημάτων (General Pickup and Delivery Problems) με δυνατότητα μεταβολής των παραμέτρων του προβλήματος κατά τη διάρκεια της κίνησης των οχημάτων. Επίσης, προσπάθεια θα καταβληθεί για την οπτικοποίηση των προτεινόμενων λύσεων επί χαρτών.

Λέξεις Κλειδιά: ανάπτυξη εφαρμογών, πρόβλημα δρομολόγησης, εφοδιαστική αλυσίδα, διπλωματική εργασία

Abstract

The purpose of this work is to develop web application to solve the problem of vehicle routing, in support of supply chain processes, with Java EE technology. The educational objective is to familiarize and explore the technologies provided by the Java EE version and to study the modern trends in the development of web applications.

Taking into account a set of vehicles, a set of refueling stations and a set of seats that need service (Pick up / Delivery) as well as the distance between the vehicles and the seats, a web application will be developed from which to allocate seats in vehicles per order to minimize the total distance of the vehicle to be driven. To solve the problem, ready-made open source libraries will be deployed in General Pickup and Delivery Problems with the ability to change the problem parameters during vehicle traffic. An effort will also be made to visualize the proposed solutions on maps.

Keywords: web development, vehicle routing, supply chain, master thesis

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε στα πλαίσια του μεταπτυχιακού προγράμματος του τμήματος Εφαρμοσμένης Πληροφορικής .

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέπων καθηγητή μου κ. Χατζηγεωργίου Αλέξανδρο για την άριστη συνεργασία που είχα μαζί του και την άμεση ανταπόκρισή του.

Ακόμη, δεν θα μπορούσα να μην ευχαριστήσω την οικογένειά μου, η οποία με βοηθά και με στηρίζει όλα αυτά τα χρόνια και τον φίλο και συνάδελφο Γαϊτανίδη Βασίλειο.

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

Στο παρόν κεφάλαιο γίνεται μια μικρή παρουσίαση του αντικειμένου της εργασίας, της δομής αυτού και των σκοπών του. Ακόμη, παρουσιάζεται η σύνοψη των επόμενων κεφαλαίων ώστε να μπορέσει ο αναγνώστης να κατατοπιστεί επαρκώς σχετικά με το τι πρόκειται να διαβάσει στις επόμενες σελίδες.

1.1 Αντικείμενο διπλωματικής εργασίας

Αντικείμενο της εργασίας αυτής αποτελεί η ανάπτυξη μίας web εφαρμογής για την επίλυση του προβλήματος της δρομολόγησης οχημάτων, στο πλαίσιο της υποστήριξης των διαδικασιών της εφοδιαστικής αλυσίδας, με την τεχνολογία Java EE. Στόχος της εργασίας είναι η εξοικείωση και η διερεύνηση των τεχνολογιών που παρέχονται από την έκδοση της Java EE και η μελέτη των σύγχρονων τάσεων στην ανάπτυξη διαδικτυακών εφαρμογών.

Το πρόβλημα προς επίλυση είναι γνωστό στη βιβλιογραφία ως Vehicle Routing και παρουσιάστηκε αρχικά από τους Dantzig & Ramser το 1959 (G. B. Dantzig, 1959). Στο Vehicle Routing υπάρχουν ως δεδομένα ένα σύνολο οχημάτων, ένα σύνολο σταθμών ανεφοδιασμού και ένα σύνολο θέσεων που χρειάζονται εξυπηρέτηση (Pick up/Delivery) καθώς και η απόσταση των οχημάτων και των θέσεων και ζητείται η κατανομή θέσεων σε οχήματα ανά παραγγελία ώστε να ελαχιστοποιείται η συνολική απόσταση των οχημάτων που πρέπει να διανυθεί.

Για την επίλυση του προβλήματος έγινε χρήση έτοιμων βιβλιοθηκών ανοιχτού κώδικα σε Java προσανατολισμένων στη λύση αντίστοιχων προβλημάτων (General Pickup & Delivery Problems) με δυνατότητα μεταβολής των παραμέτρων του προβλήματος. Επίσης, έγινε προσπάθεια για την οπτικοποίηση των προτεινόμενων λύσεων επί χαρτών.

1.2 Διάρθρωση της μελέτης

Η παρούσα εργασία αποτελείται από πέντε κεφάλαια. Στο πρώτο κεφάλαιο περιγράφεται το αντικείμενο της εργασίας. Στο δεύτερο κεφάλαιο πραγματοποιείται μία σύντομη παρουσίαση της τεχνολογιών και των βιβλιοθηκών που χρησιμοποιήθηκαν για την υλοποίηση της εργασίας.

Στο τρίτο κεφάλαιο πραγματοποιείται μία παρουσίαση των τεχνικών επίλυσης παρόμοιων προβλημάτων. Στο τέταρτο κεφάλαιο γίνεται εκτενής περιγραφή της υλοποίησης της εφαρμογής.

Στο πέμπτο κεφάλαιο παρουσιάζεται αναλυτικά η λειτουργικότητα της εφαρμογής. Στο έκτο και τελευταίο κεφάλαιο, καταγράφονται τα συμπεράσματα της παρούσας εργασίας.

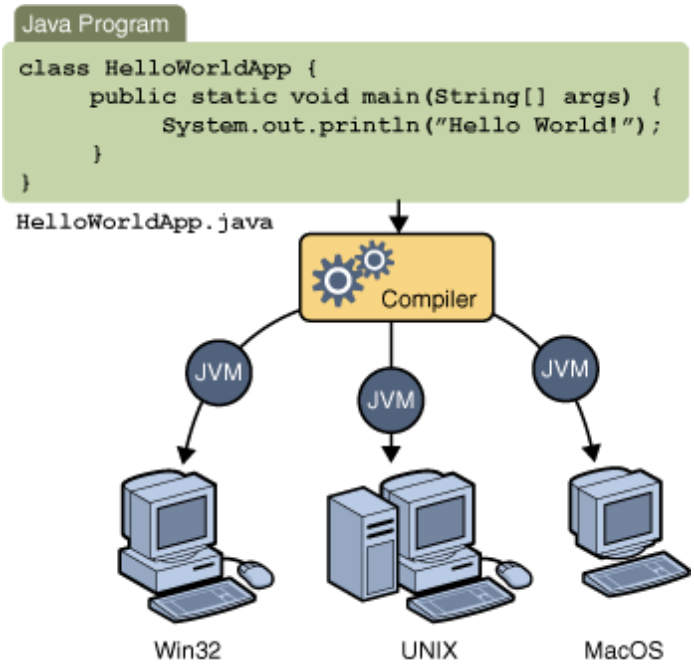
Κεφάλαιο 2

Τεχνολογίες ανάπτυξης εφαρμογής

Στο παρόν κεφάλαιο γίνεται μια μικρή παρουσίαση των τεχνολογιών που χρησιμοποιήθηκαν κατά τη διάρκεια της ανάπτυξης της εφαρμογής. Πιο συγκεκριμένα, παρουσιάζεται η γλώσσα προγραμματισμού Java, η πλατφόρμα Java EE, το Spring framework, η βιβλιοθήκη Optaplanner, το GraphHopper Route Optimization API καθώς και το Maven.

2.1 Java

Η Java είναι μια αντικειμενοστρεφής γλώσσα προγραμματισμού γενικού σκοπού και δημιουργήθηκε από τον J. Gosling κ.α. στη Sun Microsystems το 1991 και αρχικά ονομαζόταν Oak. Η Java έχει σχεδιαστεί έτσι ώστε τα εκτελέσιμα της να μπορούν να τρέχουν σε οποιαδήποτε πλατφόρμα χωρίς επιπλέον μεταγλώττιση (Oracle, Oracle, 2017). Τα αρχεία πηγαίου κώδικα Java μεταγλωττίζονται σε αρχεία bytecode και όχι σε κώδικα μηχανής, τα οποία είναι ανεξάρτητα μηχανής και μπορούν να εκτελεστούν από μία Java Virtual Machine (JVM). Ο τρόπος αυτός λειτουργίας παρουσιάζεται στην Εικόνα 1. Η JVM φορτώνει τις κλάσεις που χρειάζονται για να εκτελεστεί το Java πρόγραμμα και στη συνέχεια επικυρώνει την εγκυρότητα των bytecode αρχείων πριν τα εκτελέσει.



Εικόνα 1: Μεταφερσιμότητα της Java.

Η πλατφόρμα της Java αποτελείται από δύο μέρη, τη Java Virtual Machine (JVM) και τα Java Application Programming Interfaces (APIs). Τα APIs είναι βιβλιοθήκες που παρέχουν βασικές συναρτήσεις έτοιμες για χρήσεις και υπάρχουν τέσσερις εκδόσεις:

- Java Card για τις smartcards.
- Java Micro Edition (Java ME) για περιβάλλοντα με περιορισμένους πόρους, όπως κινητές συσκευές.
- Java Standard Edition (Java SE) για τοπικές εφαρμογές που τρέχει τοπικά σε έναν υπολογιστή.
- Java Enterprise Edition (Java EE) για μεγάλα κατανεμημένα εταιρικά ή διαδικτυακά περιβάλλοντα.

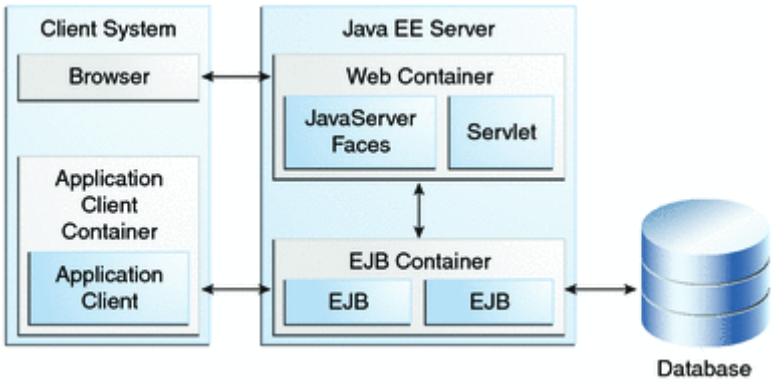
Στη Java υπάρχουν οι πρωτογενείς τύποι δεδομένων: Boolean, byte, short, int, long, float, double, char. Εκτός των πρωτογενών τύπων δεδομένων, όλα τα υπόλοιπα είναι αντικείμενα της κλάσης java.lang.Object και δημιουργούνται με τη λέξη new. Για παράδειγμα για να οριστεί ένα αλφαριθμητικό, πρέπει να γραφτεί «String s = new String(“Hello world”);».

Η διαχείριση μνήμης πραγματοποιείται αυτόματα μέσω του νήματος garbage collector, χωρίς να χρειάζεται να ασχοληθεί ο χρήστης με αυτό. Τα Java Applets είναι διαδικτυακές εφαρμογές, που τοποθετούνται σε ιστοσελίδες και εκτελούνται από τη JVM ενός φυλλομετρητή.

2.2 Java EE

Η πλατφόρμα Java Enterprise Edition (Java EE) αποτελεί ένα σύνολο προδιαγραφών (specifications), που επεκτείνουν τη Java Standard Edition (Java SE) στοχεύοντας σε κατανεμημένα υπολογιστικά συστήματα και υπηρεσίες διαδικτύου (Oracle, Oracle, 2017). Τα specifications ορίζουν Application Programming Interfaces (APIs) και την αλληλεπίδραση που έχουν με άλλες υπηρεσίες. Η Java EE χρησιμοποιείται συχνά σε υπηρεσίες ηλεκτρονικού εμπορίου, λογιστικές υπηρεσίες, καθώς και σε τραπεζικά πληροφοριακά συστήματα. Η πιο πρόσφατη έκδοση της πλατφόρμας είναι η Java EE 8 και κυκλοφόρησε στις 31 Αυγούστου του 2017 (Oracle, Github, 2017).

Η Java EE περιέχει διάφορα specifications που εξυπηρετούν διαφορετικούς σκοπούς, όπως η δημιουργία ιστοσελίδων και ανάγνωση/εγγραφή σε βάση δεδομένων. Τα specifications μπορούν να διαχωριστούν σε τέσσερις κατηγορίες, στα web specifications, στα web service specifications, στα enterprise specifications και στα λοιπά specifications. Μια επισκόπηση των βασικών στοιχείων της πλατφόρμας Java EE παρουσιάζεται στην Εικόνα 2.



Εικόνα 2: Java EE Platform.

Τα *web specifications* της Java EE είναι τα ακόλουθα:

- Servlet: ορίζει τη διαχείριση των αιτημάτων τύπου HTTP σε σύγχρονο ή ασύγχρονο τρόπο. Ανήκει στο χαμηλό επίπεδο και άλλα specifications της Java EE βασίζονται σε αυτό.
- WebSocket: Ορίζει ένα σύνολο από APIs για την εξυπηρέτηση συνδέσεων τύπου WebSocket (πρωτόκολλο τηλεπικοινωνιών σχετικό με σύνδεση τύπου TCP).
- Java Server Faces (JSF): μία τεχνολογία για την κατασκευή διεπαφών χρήστη.
- Unified Expression Language (EL): Μία απλή γλώσσα σχεδιασμένη για την ικανοποίηση συγκεκριμένων αναγκών των προγραμματιστών που αναπτύσσουν διαδικτυακές εφαρμογές. Χρησιμοποιείται συνήθως στη JSF, αλλά μπορεί να χρησιμοποιηθεί σε όλη την πλατφόρμα.

Τα *web service specifications* της Java EE είναι τα ακόλουθα:

- Java API για υπηρεσίες διαδικτύου τύπου REST.
- Java API για την επεξεργασία πληροφοριών σε μορφή JSON.
- Java API για τη μετατροπή πληροφοριών τύπου JSON σε κλάσεις Java και το αντίστροφο.
- Αρχιτεκτονική που επιτρέπει την αντιστοίχιση XML σε αντικείμενα Java.
- Java API για τη δημιουργία διαδικτυακών υπηρεσιών τύπου SOAP.

Τα *enterprise specifications* της Java EE είναι τα ακόλουθα:

- Εισαγωγή περιεχομένου και εξαρτήσεων.
- Enterprise JavaBean (EJB): ένα σύνολο APIs, το οποίο υποστηρίζει ένα αντικείμενο τύπου EJB container για την παροχή μεταφορών, απομακρυσμένες κλήσεις συναρτήσεων κ.λ.π. από αντικείμενα επιχειρήσεων.
- Java Persistence API: προδιαγραφές για την αντιστοίχιση μεταξύ κλάσεων Java και πίνακες βάσης δεδομένων.
- Java Transaction API: Περιέχει τις διεπαφές και τους συμβολισμούς για την αλληλεπίδραση με την υποστήριξη μεταφορών που προσφέρει η Java EE.
- Java Message Service: επιτρέπει στις εφαρμογές Java να δημιουργήσουν, να στείλουν, να λάβουν και διαβάσουν μηνύματα που προέρχονται από το σύστημα της επιχείρησης.

Τα *λοιπά specifications* της Java EE είναι τα ακόλουθα:

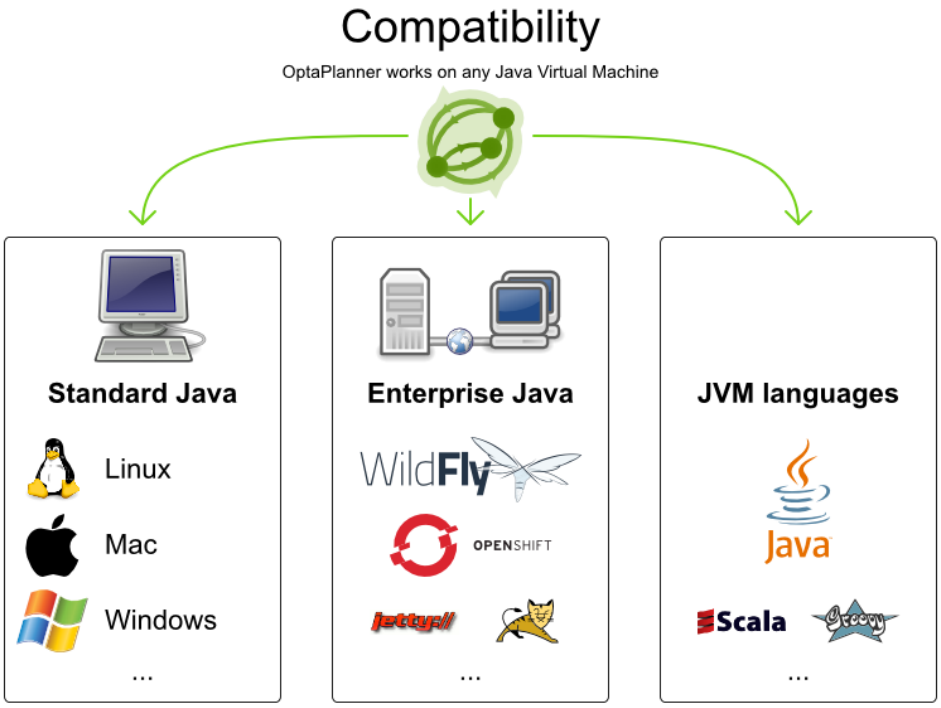
- Validation: περιέχει τους συμβολισμούς και τις διεπαφές για την υπηρεσία επαλήθευσης που προσφέρεται από το API Bean Validation.
- Batch Applications: προσφέρεται η δυνατότητα να τρέχουν στο παρασκήνιο εφαρμογές που απαιτούν μεγάλο υπολογιστικό χρόνο και μπορεί να εκτελούνται περιοδικά.
- Αρχιτεκτονική Java EE Connector: αφορά τη σύνδεση εφαρμογών εξυπηρετητή και πληροφοριακά συστήματα επιχειρήσεων.

2.3 Βιβλιοθήκη OptaPlanner

Η βιβλιοθήκη OptaPlanner αποτελεί ένα λύτη προβλημάτων ικανοποίησης περιορισμών (constraint specification solver) (OptaPlanner, 2017). Η OptaPlanner βελτιστοποιεί το σχεδιασμό πόρων μίας επιχείρησης σε διάφορες περιπτώσεις, όπως η Δρομολόγηση Οχημάτων (Vehicle Routing), η Ανάθεση Βάρδιας σε Εργαζόμενους (Employee Rostering), Βελτιστοποίηση Cloud (Cloud Optimization) κ.α. Κάθε επιχείρηση αντιμετωπίζει ένα πρόβλημα χρονοπρογραμματισμού που έχει περιορισμένους πόρους (π.χ. πλήθος εργαζόμενων, χρόνος, χρήμα) και σαν στόχο να παρέχει προϊόντα ή υπηρεσίες σε πελάτες. Η βιβλιοθήκη OptaPlanner παρέχει τη βέλτιστη λύση σε αυτά τα προβλήματα χρονοπρογραμματισμού ώστε να παράγεται το μέγιστο δυνατό έργο με τις λιγότερους δυνατούς πόρους.

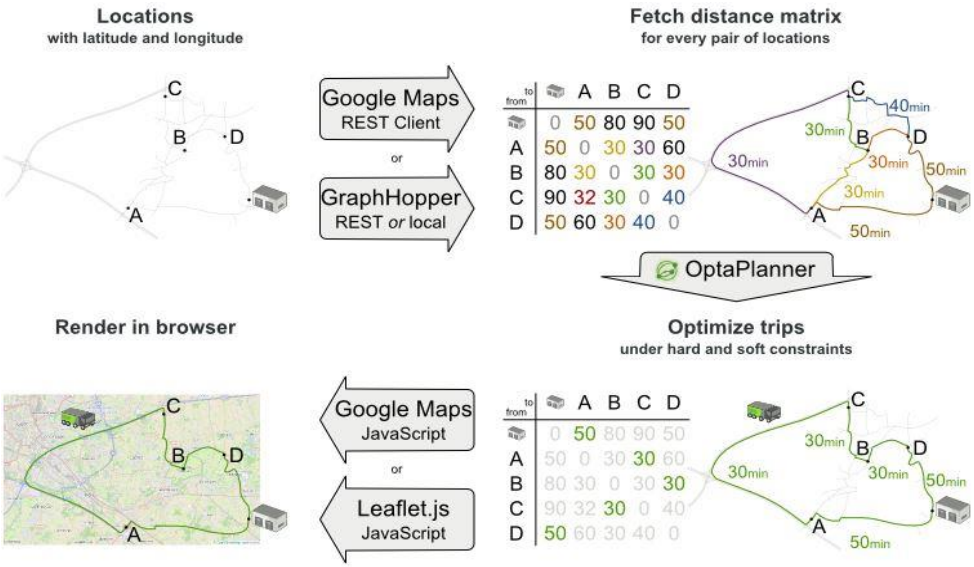
Η βιβλιοθήκη OptaPlanner είναι μια ελαφριά ενσωματωμένη μηχανή χρονοπρογραμματισμού, που επιτρέπει στους προγραμματιστές που αναπτύσσουν εφαρμογές σε Java να λύνουν προβλήματα βελτιστοποίησης αποτελεσματικά. Οι περιορισμοί εφαρμόζονται σε απλά αντικείμενα και μπορεί να επαναχρησιμοποιηθεί υπάρχων κώδικας, χωρίς να υπάρχει ανάγκη για εισαγωγή πολύπλοκων μαθηματικών εξισώσεων.

Η βιβλιοθήκη OptaPlanner είναι ανοιχτού κώδικα κι έχει γραφτεί σε Java. Μπορεί να ενσωματωθεί εύκολα σε άλλες τεχνολογίες Java, ενώ υπάρχει και στο κεντρικό repository του Maven. Η συμβατότητα της Optaplanner παρουσιάζεται στην Εικόνα 3.



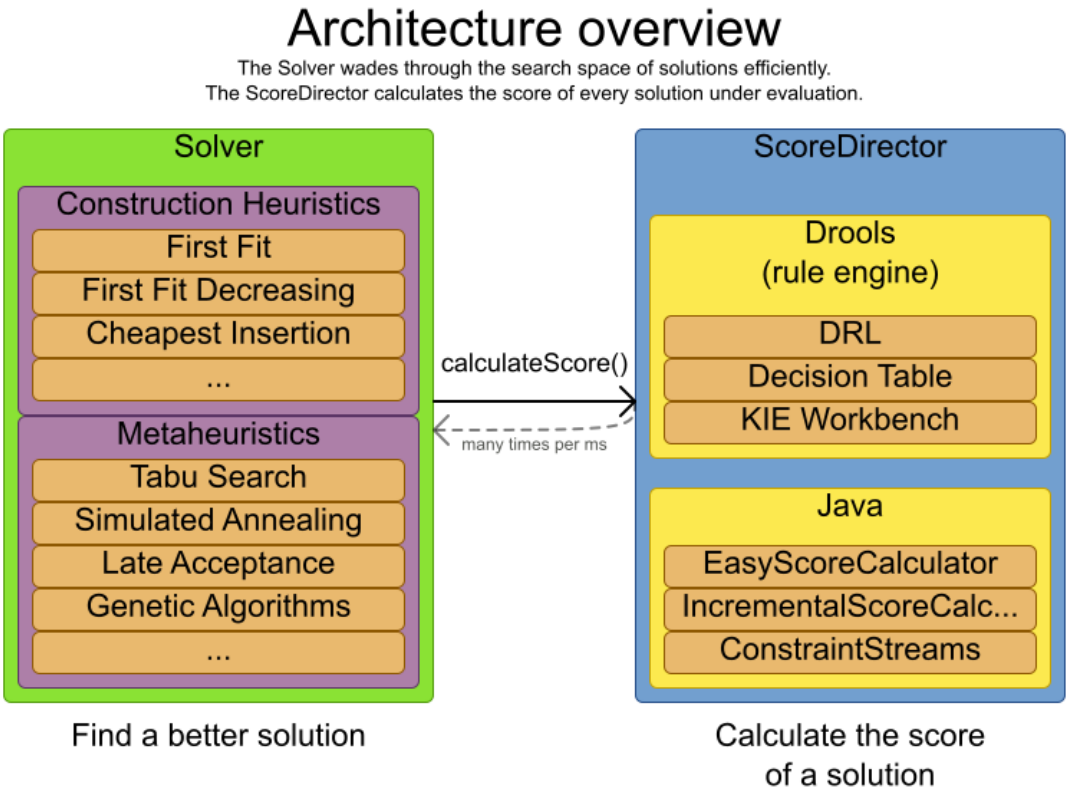
Εικόνα 3: Συμβατότητα της OptaPlanner.

Στην Εικόνα 4 φαίνεται πώς γίνεται η υλοποίηση της OptaPlanner με τους Χάρτες Google ή το GraphHopper (OpenStreetMap) που θα χρησιμοποιήσουμε



Εικόνα 4 : Ενσωμάτωση της OptaPlanner σε χάρτες

Για τον υπολογισμό των λύσεων η OptaPlanner αρχικά χρησιμοποιεί μία παραλλαγή των ευρετικών μεθόδων (constructive heuristic) όπου δημιουργείται μία άδεια λύση και επανειλημμένα επεκτείνει την συγκεκριμένη λύση έως ότου ολοκληρωθεί. Στην συνέχεια εφαρμόζεται κάποια από τις ευρετικές μεθόδους αναζήτησης, στην περίπτωσή μας Tabu Search υπολογίζοντας το καλύτερο score για να καταλήξει στην βέλτιστη λύση. Στην Εικόνα 5 βλέπουμε πως αντιμετωπίζεται το πρόβλημα από την βιβλιοθήκη.



Εικόνα 5: Αρχιτεκτονική υπολογισμού λύσεων στην OptaPlanner

2.4 Spring Framework

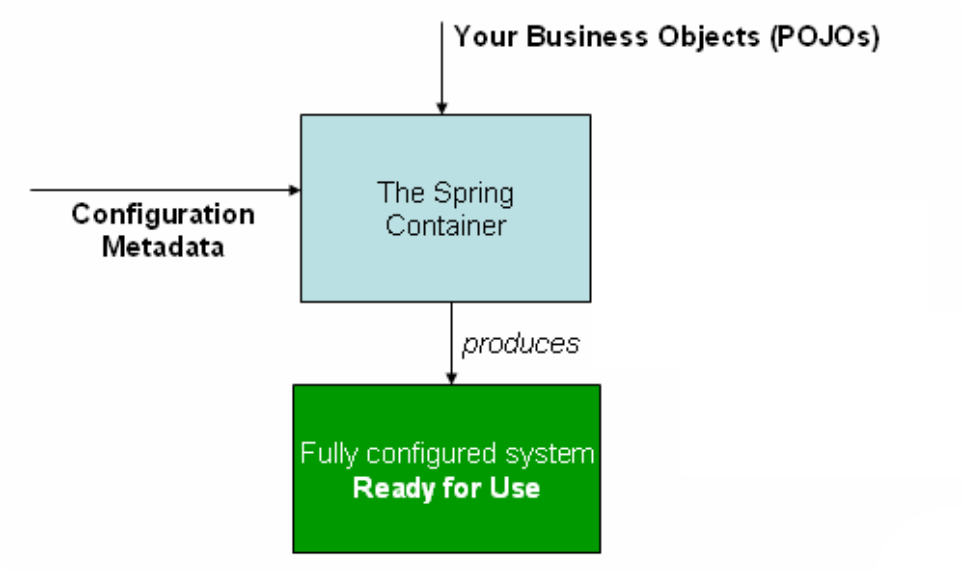
Το Spring Framework είναι ένα framework εφαρμογής και αντιστροφής του ελέγχου στην πλατφόρμα της Java. Οι κεντρικές λειτουργίες του Spring μπορούν να χρησιμοποιηθούν από οποιαδήποτε εφαρμογή Java, αλλά υπάρχουν επεκτάσεις που μπορούν να χρησιμοποιηθούν για τη δημιουργία διαδικτυακών εφαρμογών πάνω από την πλατφόρμα Java EE. Το Spring είναι ανοιχτού κώδικα κι έχει γίνει εξαιρετικά δημοφιλές στην κοινότητα της Java σαν προσθήκη ή ακόμα και αντικαταστάτης του EJB.

Το Spring παρέχει ένα συνοπτικό μοντέλο προγραμματισμού και διαχείρισης για τις μοντέρνες εφαρμογές που δημιουργούνται με Java. Ένα σημαντικό στοιχείο του είναι η υποστήριξη δομών στο επίπεδο της εφαρμογής, καθώς βοηθάει τις ομάδες ανάπτυξης λογισμικού να εστιάζουν στην επιχειρησιακή λογική σε επίπεδο εφαρμογής, χωρίς να αναλώνονται στο γράψιμο κώδικα διαχείρισης (Spring, 2018).

Αυτό επιτυγχάνεται με τα Spring Modules, τα οποία αποτελούνται από πολλά τμήματα κάθε ένα από τα οποία παρέχει ένα συγκεκριμένο σύνολο από λειτουργίες και μπορεί να χρησιμοποιηθεί ανεξάρτητα από τα υπόλοιπα. Επιπλέον, το Spring παρέχει επεκτασιμότητα καθώς διαθέτει σημεία ενσωμάτωσης για την εύκολη συνεργασία με άλλα frameworks και APIs (Dhrubojyoti, 2008)

Ο πυρήνας πάνω στον οποίο βασίζεται το Spring και όλα τα υπόλοιπα τμήματα ονομάζεται Core Container και Inversion of Control (IoC) Container. Τα αντικείμενα που διαμορφώνουν το σκελετό μιας Spring εφαρμογής και διαχειρίζονται από τον IoC αναφέρονται ως beans. Ένα bean είναι ένα απλό αντικείμενο (Plain Old Java Onject - POJO) που αρχικοποιείται και επεξεργάζεται από τον IoC. Τα beans,οι μεταξύ τους εξαρτήσεις και τα χαρακτηριστικά τους αναπαρίστανται στα configuration metadata που παρέχονται μέσω xml αρχείων και χρησιμοποιούνται από τον IoC.

Στην Εικόνα 6 παρουσιάζεται ένα διάγραμμα που εξηγεί πως λειτουργεί το Spring στο υψηλό επίπεδο. Οι κλάσεις της εφαρμογής συνδυάζονται με τα metadata διαχείρισης, ώστε αφού το ApplicationContext δημιουργηθεί και αρχικοποιηθεί, να υπάρχει ένα σύστημα ρυθμισμένο κι έτοιμο για εκτέλεση.



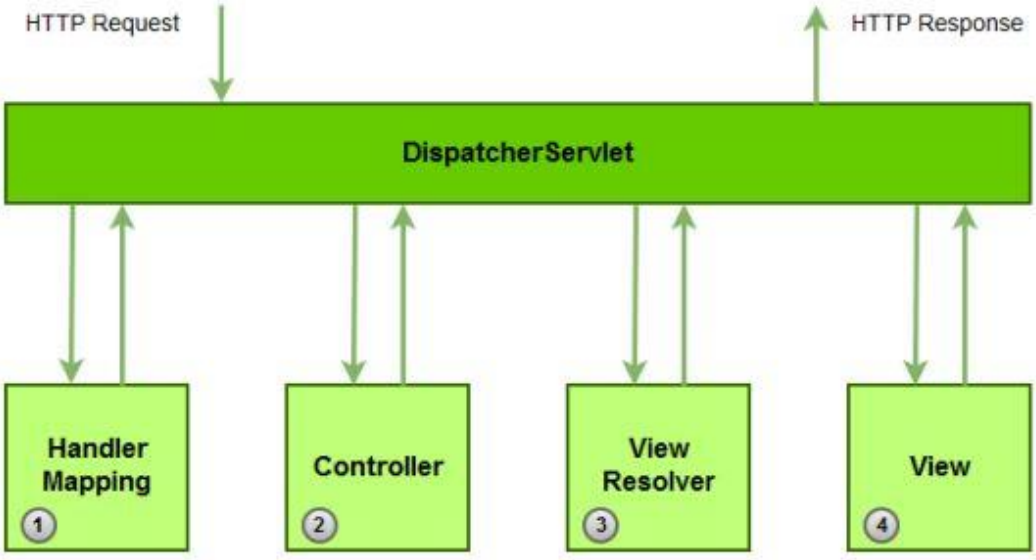
Εικόνα 6: Μοντέλο Spring.

Οι κύριες λειτουργίες του Spring είναι οι ακόλουθες:

- Τεχνολογίες πυρήνα: dependency injection, events, resources, i18n, validation, data binding, μετατροπή τύπων, SpEL, AOP.
- Έλεγχος: αντικείμενα mock, TestContext framework, Spring MVC Test, WebTestClient.
- Πρόσβαση δεδομένων: transactions, DAO support, JDBC, ORM, Marshalling XML.
- Τα διαδικτυακά framework Spring MVC και Spring WebFlux.
- Ενσωμάτωση: remoting, JMS, JCA, JMX, e-mail, tasks, cache, χρονοπρογραμματισμός.
- Γλώσσες: Kotlin, Groovy, δυναμικές γλώσσες.

Το interface org.springframework.context.ApplicationContext παίζει τον ρόλο του Container και το implementation του interface αυτού είναι το WebApplicationContext. Το κεντρικό servlet (το οποίο κάνει extend το HttpServlet) πάνω στο οποίο είναι σχεδιασμένο το Spring Web MVC Framework είναι το DispatcherServlet. Το DispatcherServlet αποτελεί έναν front controller ο οποίος αναλαμβάνει την παραλαβή των requests και την προώθησή τους σε άλλους Controllers και Views. Σε μία Spring MVC εφαρμογή ο DispatcherServlet

χρησιμοποιεί Beans (Controllers, Handlers, Resolvers , Services, DAOs κτλ.) του Spring Framework για να επεξεργαστεί τα requests και να τα περάσει στα αντίστοιχα views. Η πορεία επεξεργασίας των αιτημάτων (requests) που δέχεται ο DispatcherServlet του Spring παριστάνεται στην Εικόνα 7: HTTP Request στο Spring και αποτελείται από τα εξής βήματα (Johnson, 2004). Όταν φτάσει ένα HTTP request στον DispatcherServlet, αυτός συμβουλεύεται τον HandlerMapping έτσι ώστε να καλέσει τον σωστό Controller. Ο Controller στη συνέχεια παίρνει το αίτημα (request) και καλεί το κατάλληλο service το οποίο τροποποιεί το μοντέλο και επιστρέφει το όνομα του view στον DispatcherServlet, ο οποίος με τη βοήθεια του ViewResolver επιλέγει το κατάλληλο view και αφού του "δώσει" τα δεδομένα του model, το view αυτό εμφανίζεται στον browser.



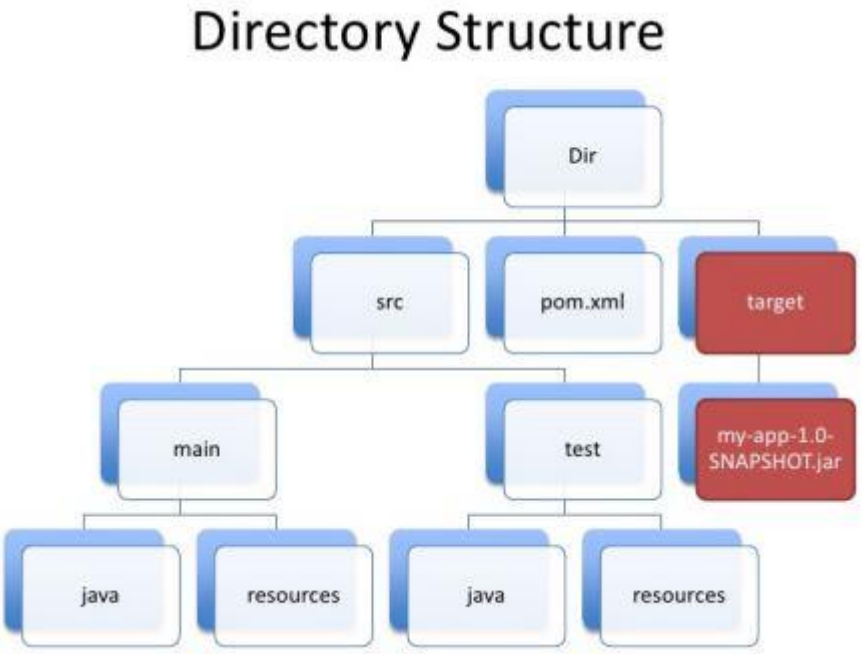
Εικόνα 7: HTTP Request στο Spring

2.5 Maven

Το Maven είναι ένα εργαλείο που χρησιμοποιείται για την ανάπτυξη και την διαχείριση κατά κύριο λόγο των Java projects. Κύριος στόχος του Maven είναι να περιγράψει με εύκολο τρόπο την δομή και τα dependencies του λογισμικού. Μέσω ενός xml αρχείου που ονομάζεται pom (Project Object Model), στο οποίο

περιγράφονται τα στοιχεία του project όπως για παράδειγμα ο τρόπος που θα γίνει το build, οι επεκτάσεις (plugins) που θα χρησιμοποιηθούν , οι βιβλιοθήκες της Java κ.α.,

Το Maven κατεβάζει δυναμικά τις βιβλιοθήκες Java και τις επεκτάσεις από ένα ή περισσότερα repositories όπως το Maven 2 Central Repository και τα αποθηκεύει σε μια τοπική μνήμη cache(.m2). Σε αυτή υπάρχουν όλες οι βιβλιοθήκες που είναι απαραίτητες για να τρέξει το project. (Miller, 2010). Τα maven projects έχουν την παρακάτω δομή:



Εικόνα 8: Δομή maven project

Όνομα φακέλου	**Σκοπός**
project home	Περιέχει το pom.xml και τους υποφακέλους
src/main/java	Περιέχει τον παραδοτέο πηγαίο Java κώδικα για το project
src/main/resources	Περιέχει τα παραδοτέα resources αρχεία για το έργο(π.χ. property αρχεία)
src/test/java	Περιέχει τον πηγαίο κώδικα test Java (π.χ test JUnit) για το project
src/test/resources	Περιέχει τα απαραίτητα resources για το testing του project

Πίνακας 1: Δομή maven project

2.6 GraphHopper Route Optimization API

Το GraphHopper Route Optimization API χρησιμοποιήθηκε για τον ακριβή υπολογισμό και την απεικόνιση των δρομολογίων πάνω στους χάρτες. Είναι μια open-source βιβλιοθήκη δρομολόγησης και ένας server υλοποιημένος σε Java και παρέχει ένα interface που ονομάζεται GraphHopper Maps καθώς επίσης και ένα API δρομολόγησης μέσω HTTP.

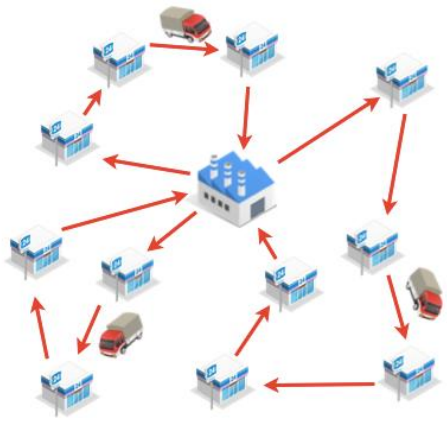
Η εφαρμογή είναι σχεδιασμένη να μπορεί να τρέχει σε server, desktop, Android, iOS ή στο Raspberry Pi. Κατά κύριο λόγο χρησιμοποιεί τα δεδομένα του OpenStreetMap για το οδικό δίκτυο καθώς και υψομετρικά δεδομένα από το Shuttle Radar Topography Mission. (Route Optimization API, n.d.)

Ο πηγαίος κώδικας του GraphHopper υπάρχει στο github. Περιλαμβάνει ολόκληρο το project, την Android εφαρμογή καθώς και server για να μπορεί ο καθένας να τρέχει την εφαρμογή τοπικά στο μηχάνημά του. Διαθέτει Apache License που επιτρέπει σε όλους να μπορούν να το προσαρμόσουν και να το ενσωματώσουν σε ελεύθερα ή εμπορικά προϊόντα, κάνοντάς το έτσι μια από τις πιθανές εναλλακτικές λύσεις στα ήδη υπάρχοντα προϊόντα.

Κεφάλαιο 3

Vehicle Routing

Το πρόβλημα Δρομολόγησης Οχημάτων (Vehicle Routing) είναι ένα συνδυαστικό πρόβλημα βελτιστοποίησης και μπορεί να διατυπωθεί ως εξής: «Ποιο είναι το βέλτιστο σύνολο διαδρομών που πρέπει να διανύσει μία ομάδα οχημάτων ώστε να παραδώσει προϊόντα σε ένα συγκεκριμένο πλήθος πελατών;». Μία γραφική απεικόνιση του προβλήματος φαίνεται στην Εικόνα 9.



Εικόνα 9: Πρόβλημα Vehicle Routing.

Το πρόβλημα Vehicle Routing αποτελεί γενίκευση του γνωστού προβλήματος Travelling Salesman Problem (TSP). Αρχικά, το Vehicle Routing παρουσιάστηκε στο (G. B. Dantzig, 1959) όπου παρουσιάστηκε η πρώτη αλγοριθμική προσέγγιση και εφαρμόστηκε στις παραδόσεις των πετρελαϊκών εταιριών. Το 1964 οι Clarke και Wright βελτίωσαν την αρχική προσέγγιση με μία αποδοτική άπληστη μέθοδο, που ονομάστηκε αλγόριθμος savings.

Το οδικό δίκτυο μπορεί να απεικονιστεί με ένα γράφο όπου οι ακμές είναι οι δρόμοι και οι κορυφές είναι οι διασταυρώσεις μεταξύ των δρόμων. Οι ακμές μπορεί να είναι κατευθυνόμενες ή μη ανάλογα με το τι επιτρέπεται στην κυκλοφορία του κάθε δρόμου. Κάθε ακμή σχετίζεται με ένα κόστος το οποίο συνήθως είναι το μήκος του ή ο χρόνος που χρειάζεται για να το διανύσει το όχημα. Η επιλογή του κόστους μπορεί να σχετίζεται με το είδος του οχήματος. Για να υπολογιστεί το συνολικό κόστος κάθε διαδρομής, θα πρέπει να είναι γνωστό το κόστος και ο χρόνος της

διαδρομής μεταξύ κάθε πελάτη και των σταθμών. Έτσι, ο αρχικός γράφος μετασχηματίζεται σε γράφο όπου οι κορυφές είναι τα σημεία παράδοσης (πελάτες) και οι σταθμοί, ενώ οι ακμές είναι οι δρόμοι που τους ενώνουν. Το κόστος κάθε ακμής είναι το ελάχιστο κόστος μεταξύ των δύο σημείων στον αρχικό γράφο. Η διαδικασία αυτή πραγματοποιείται με αλγόριθμους επίλυσης του προβλήματος εύρεσης του συντομότερου μονοπατιού και το αποτέλεσμα είναι ένας πλήρης γράφος.

Σε ορισμένες περιπτώσεις είναι αδύνατον να ικανοποιηθούν οι απαιτήσεις όλων των πελατών με αποτέλεσμα τα εργαλεία επίλυσης να μειώνουν τις απαιτήσεις ορισμένων πελατών ή να αφήνουν κάποιους πελάτες χωρίς εξυπηρέτηση. Για την αντιμετώπιση τέτοιων καταστάσεων μπορεί να εισαχθεί μία μεταβλητή προτεραιότητας για κάθε πελάτη ή να επιβληθούν κάποιες ποινές για μερική ή ολική έλλειψη εξυπηρέτησης για κάθε πελάτη.

Ο καθορισμός της βέλτιστης λύσης του προβλήματος Vehicle Routing είναι NP-hard (Toth, 2001), συνεπώς το μέγεθος των προβλημάτων που μπορούν να επιλυθούν κατά το βέλτιστο τρόπο χρησιμοποιώντας μαθηματικό προγραμματισμό μπορεί να είναι περιορισμένο. Συνεπώς, τα εμπορικά εργαλεία επίλυσης χρησιμοποιούν ευρετικές μεθόδους για την επίλυση ρεαλιστικών προβλημάτων των οποίων το μέγεθος είναι μεγάλο, καθώς και η συχνότητα ανάγκης για επίλυση αυξημένη.

Το πρόβλημα Vehicle Routing έχει πολλές προφανείς εφαρμογές στη βιομηχανία. Η χρήση υπολογιστικών προγραμμάτων που βελτιστοποιούν τις παραδόσεις μπορούν να οδηγήσει έως και μείωση κατά 5% το κόστους των παραδόσεων μίας εταιρίας. Το ποσοστό αυτό είναι σημαντικό, καθώς ο τομέας μεταφορών μπορεί να φτάνει έως και το 10% του Ακαθάριστου Εγχώριου Προϊόντος.

Στο πλαίσιο της παρούσας εργασίας θα αντιμετωπιστεί μια παραλλαγή του βασικού προβλήματος, η οποία ονομάζεται Vehicle Routing Problem with Pickup and Delivery. Στην παραλλαγή αυτή υπάρχει ένα πλήθος αγαθών που χρειάζεται να μετακινηθούν από προκαθορισμένα σημεία παραλαβής σε διαφορετικά ορισμένα σημεία παράδοσης με χρήση ενός ετερογενούς στόλου οχημάτων. Ο στόχος είναι να βρεθούν οι βέλτιστες διαδρομές για την ομάδα των οχημάτων ώστε να επισκεφτούν τα σημεία παραλαβής και παράδοσης.

Το πρόβλημα Vehicle Routing Problem with Pickup and Delivery περιέχει επιπλέον περιορισμούς. Κάθε όχημα πρέπει να επισκεφτεί την κάθε τοποθεσία μία φορά, χωρίς να παραβιάζεται η χωρητικότητα των οχημάτων. Επίσης υπάρχουν τερματικοί σταθμοί από όπου ξεκινάει και τερματίζει κάθε όχημα.

3.1 Μαθηματική Διατύπωση του προβλήματος Vehicle Routing

Έστω ότι i=0 είναι η κεντρική αποθήκη ή το κέντρο διανομής. Τότε i=1,2,3...,n θα είναι οι πελάτες μας. Θεωρούμε ότι κάθε πελάτης i έχει ζήτηση q_i ποσότητα προϊόντων και το κόστος μετάβασης από τον πελάτη i στον j ορίζετε ως c_j. Εάν η εταιρία διαθέτει K οχήματα που εκτελούν τις μεταφορές, η χωρητικότητα κάθε οχήματος θα είναι Q_K . Επιπλέον, γίνεται η υπόθεση πως όλοι οι πελάτες και τα οχήματα θα εξυπηρετούν με φθίνουσα σειρά τα q_i και Q_K . Τέλος, σε κάθε όχημα θα αντιστοιχεί μία διαδρομή η οποία θα ξεκινάει και θα καταλήγει στο κέντρο διανομής. (Lawler, 1985).Η αντικειμενική συνάρτηση που έχουμε προς ελαχιστοποίηση θα είναι:

$$c^* = \min \sum_{V} \sum_{ij} c_{ij} x_{ij}^{v}$$

Υπό τους περιορισμούς:

$$\sum d_i x_{ij}^{v} \leq K \qquad \text{για κάθε } v=1,2,\ldots,K$$

$$X=[x_{ij}^{v}] \qquad \in S^*$$

$$x_{ij}^{v} = \begin{cases} 1, & \text{το όχημα } v \text{ χρησιμοποιεί το τόξο } (i,j) \\ 0, & \end{cases}$$

με c_{ij} συμβολίζεται η απόσταση που διανύει ένα όχημα (το κόστος της διαδρομής) προκειμένου να μεταβεί από τον πελάτη i στον πελάτη j. Επίσης τα d_i, Q, K και S * συμβολίζουν τα εξής:

d_i = ζήτηση του πελάτη i.

Q = Χωρητικότητα οχήματος.

K = Το σύνολο οχημάτων της εταιρίας.

S* = Το σύνολο όλων των M λύσεων του προβλήματος του περιπλανώμενου πωλητή.

3.2 Ακριβείς λύσεις

Υπάρχουν τρεις κυρίαρχες προσεγγίσεις μοντελοποίησης του προβλήματος Vehicle Routing:

- Μοντελοποίηση ροής οχημάτων: Γίνεται χρήση ακέραιων μεταβλητών που σχετίζεται με κάθε ακμή, μετρώντας τις φορές που ένα όχημα διασχίζει την ακμή. Συνήθως χρησιμοποιείται στο βασικό πρόβλημα Vehicle Routing και ταιριάζει στις περιπτώσεις, όπου το κόστος της λύσης μπορεί να εκφραστεί με το άθροισμα των κοστών που σχετίζεται με τις ακμές. Υπάρχουν πολλές πρακτικές εφαρμογές, στις οποίες δεν μπορεί να εφαρμοστεί αποδοτικά.
- Μοντελοποίηση ροής αγαθών: Στην περίπτωση αυτή γίνεται χρήση επιπλέον ακέραιων μεταβλητών που σχετίζονται με κάθε ακμή, οι οποίες αναπαριστούν τη ροή των αγαθών στα μονοπάτια που διασχίζουν τα οχήματα. Η μέθοδος αυτή χρησιμοποιήθηκε πρόσφατα για την εύρεση ακριβής λύσης (Toth, 2001).
- Πρόβλημα διαμέρισης συνόλου: Στην προσέγγιση αυτή υπάρχει ένας εκθετικός αριθμός δυαδικών μεταβλητών που κάθε μία σχετίζεται με ένα διαφορετικό εφικτό κύκλωμα. Το πρόβλημα Vehicle Routing μοντελοποιείται στη συνέχεια ως πρόβλημα διαμέρισης συνόλου όπου

αναζητείται η συλλογή κυκλωμάτων με το ελάχιστο κόστος που ικανοποιεί τους περιορισμούς του αρχικού προβλήματος.

3.3 Ευρετικές μέθοδοι

Εξαιτίας της δυσκολίας επίλυσης του προβλήματος σε πραγματικές περιπτώσεις που το μέγεθος του προβλήματος είναι πολύ μεγάλο, έχουν γίνει σημαντικές ερευνητικές προσπάθειες με χρήση ευρετικών μεθόδων, όπως οι γενετικοί αλγόριθμοι ,η αναζήτηση Tabu (Glover, 1986) και η προσομοιωμένη ανόπτηση (simulated annealing) (Khachaturyan, 1979). Κάποιες από τις πιο πρόσφατες ευρετικές μεθόδους που εφαρμόστηκαν στο πρόβλημα Vehicle Routing πέτυχαν λύσεις με πολύ μικρή απόκλιση από τη βέλτιστη σε στιγμιότυπα του προβλήματος που είχαν εκατοντάδες ή χιλιάδες σημεία παράδοσης (Vidal, 2014).

Οι ευρετικές μέθοδοι είναι πιο συμπαγείς, επειδή μπορούν να προσαρμοστούν αρκετά εύκολα εφαρμόζοντας επιπλέον περιορισμούς των βασικών. Συνεπώς, η εφαρμογή ευρετικών μεθόδων πολύ συχνά προτιμάται σε πραγματικές εφαρμογές μεγάλης κλίμακας που περιέχουν πολύπλοκους περιορισμούς.

- Γενετικοί Αλγόριθμοι :

Οι Γ.Α. διατηρούν έναν πληθυσμό πιθανών λύσεων, του προβλήματος που μας ενδιαφέρει, πάνω στον οποίο δουλεύουν, σε αντίθεση με άλλες μεθόδους αναζήτησης που επεξεργάζονται ένα μόνο σημείο του διαστήματος αναζήτησης. Έτσι ένας Γ.Α. πραγματοποιεί αναζήτηση σε πολλές κατευθύνσεις και υποστηρίζει καταγραφή και ανταλλαγή πληροφοριών μεταξύ αυτών των κατευθύνσεων. Ο πληθυσμός υφίσταται μια προσομοιωμένη γενετική εξέλιξη – χρησιμοποιούνται διάφοροι γενετικοί τελεστές όπως η επιλογή, η διασταύρωση και η μετάλλαξη. Σε κάθε γενιά, οι σχετικά "καλές" λύσεις αναπαράγονται, ενώ οι σχετικά "κακές" απομακρύνονται. Ο διαχωρισμός και η αποτίμηση των διαφόρων λύσεων γίνεται με την βοήθεια μιας

αντικειμενικής συνάρτησης (objective ή fitness function), η οποία παίζει το ρόλο του περιβάλλοντος μέσα στο οποίο εξελίσσεται ο πληθυσμός. Αυτή η διαδικασία συνεχίζεται έως ότου επιτευχθεί η λύση. (Λυκοθανάσης, Εισαγωγή στις Ευρετικές Μεθόδους, 2017)

Με βάση τα παραπάνω η δομή ενός γενετικού αλγόριθμου μπορεί να αναπαρασταθεί ως εξής:

```
Αρχή
  t □ 0
   αρχικοποίηση πληθυσμού P(t)
   αξιολόγηση του πληθυσμού P(t)
   Επανέλαβε όσο δεν τηρείται κάποια από τις συνθήκες τερματισμού
   Αρχή
    t □ t+1
    εφαρμογή τελεστή επιλογής και δημιουργία πληθυσμού P(t) □ P(t-1)
    εφαρμογή τελεστή διασταύρωσης στον πληθυσμό P(t)
    εφαρμογή τελεστή μετάλλαξης στον πληθυσμό P(t)
    αξιολόγηση πληθυσμού P(t)
   Τέλος
  Τέλος
```

- Αναζήτηση Tabu :

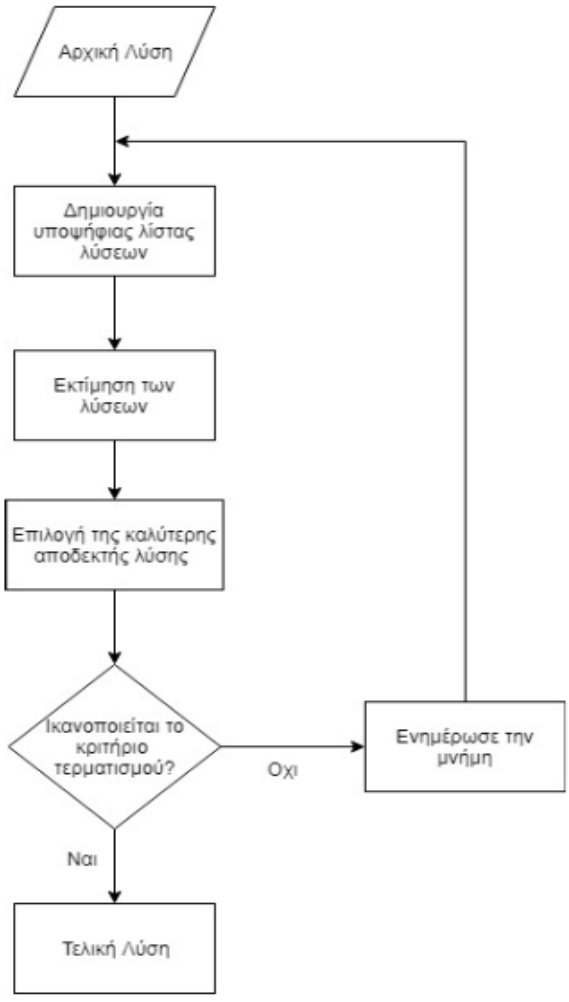
Η λογική της απαγορευμένης (taboo – tabu) έρευνας στηρίζεται στο ότι ο άνθρωπος όταν προσπαθεί να λύσει προβλήματα χρησιμοποιεί την εμπειρία του και τη μνήμη του. Το ίδιο επιχειρεί να κάνει και ο Tabu Search στο χώρο των λύσεων. Ο Tabu Search εκτελώντας μια συγκεκριμένη προσχεδιασμένη κίνηση, δημιουργεί μια γειτονιά νέων λύσεων και επιλέγει μη στοχαστικά την καλύτερη λύση της γειτονιάς ακόμα και αν αυτή είναι χειρότερη από την τρέχουσα λύση.

Έτσι, ενισχύει τη διαφοροποίηση της έρευνας και τον απεγκλωβισμό από τοπικά βέλτιστα, ενώ το μη στοχαστικό κριτήριο ενισχύει την εντατικοποίηση της έρευνας. Ταυτόχρονα, διατηρεί τη συνολικά καλύτερη λύση που έχει εξεταστεί, ως τελική λύση. Η έρευνα στηριζόμενη στο μη στοχαστικό κριτήριο, πιθανότατα να δημιουργήσει το φαινόμενο της κυκλικότητας της έρευνας, την επιστροφή δηλαδή σε μονοπάτια του χώρου των λύσεων που έχει ήδη εξετάσει.

Η λύση για την αποφυγή της κυκλικότητας είναι η εισαγωγή μιας λίστας, την Tabu List με τις N τελευταίες λύσεις που εξετάστηκαν, δηλαδή τις

«απαγορευμένες λύσεις». Αυτή η χαρακτηριστική «προσαρμοστική μνήμη» διακρίνει τον Tabu Search από άλλους αλγόριθμους τοπικής έρευνας. Η λίστα ανανεώνεται δυναμικά κατά τη διάρκεια του αλγορίθμου, με τη λογική First In – First Out. Έτσι, βοηθείται ο απεγκλωβισμός της έρευνας από ένα τοπικό βέλτιστο και η προώθηση του σε χώρους που δεν έχουν ερευνηθεί. Σκοπός της μνήμης της Tabu List είναι να ενισχύει τα χαρακτηριστικά των λύσεων που κρίθηκαν ιστορικά ισχυρά, και να αποκλείει τις αδύναμες περιοχές. Γι' αυτό, καλό θα ήταν να «απαγορεύει» στοιχεία ή λύσεις που έχουν κριθεί κατά κάποιο τρόπο πιο ακατάλληλα / αδύναμα ως τώρα, ώστε να προωθεί την έρευνα σε ανεξερεύνητους χώρους που υπόσχονται πιο ποιοτικές λύσεις (Μπαλαράς, 2016)

Παρακάτω παρουσιάζεται το διάγραμμα ροής ενός TS:



Εικόνα 10 : Διάγραμμα ροής Tabu search

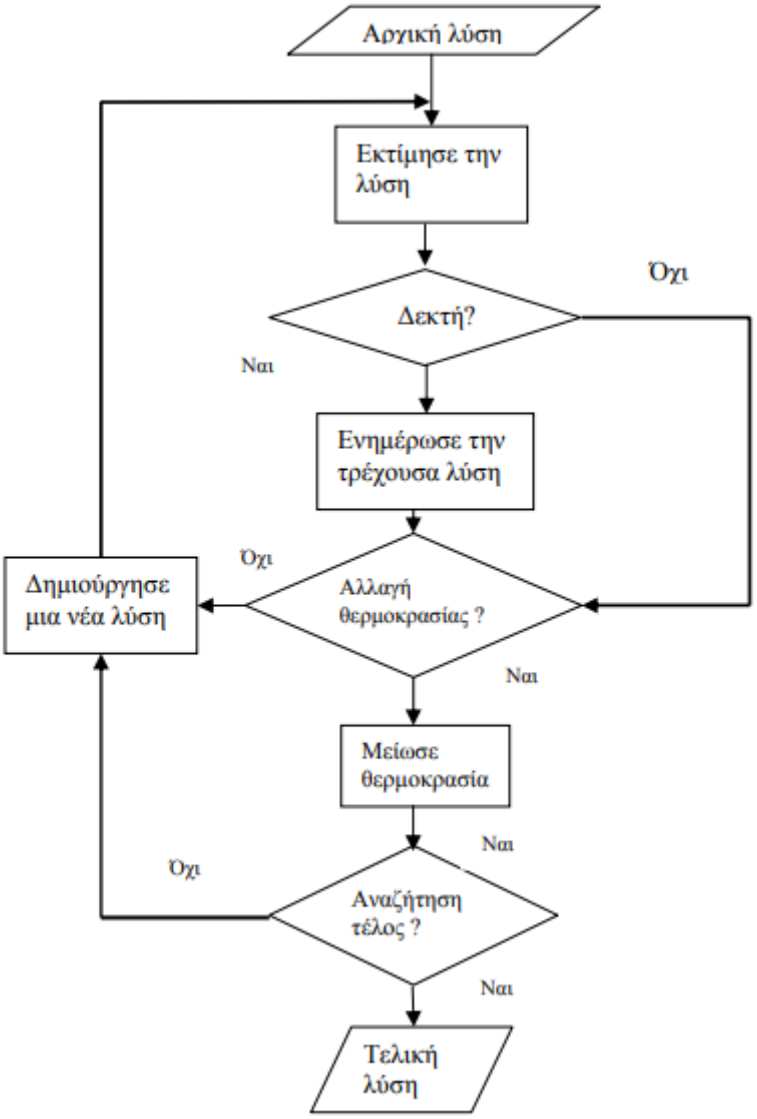
- Simulated Annealing :

Η συγκεκριμένη μέθοδος ανήκει στην γενικότερη κατηγορία μεθόδων τοπικής αναζήτησης, οι οποίες παράγουν ευρηματικό ή τυχαίο τρόπο μια αρχική λύση, την οποία στην συνέχεια βελτιώνουν επαναληπτικά. Η μέθοδος simulated annealing δέχεται λύσεις, οι οποίες απομακρύνουν την αναζήτηση από τα τοπικά βέλτιστα, καθώς η απόδοση όλων των μεθόδων τοπικής αναζήτησης εξαρτάται από την συχνότητα παγίδευσής τους σε τοπικά βέλτιστα, τα οποία συνήθως απέχουν πολύ από το ολικό βέλτιστο. Για τον λόγο αυτό χρησιμοποιείται σε κάθε βήμα k μία πιθανότητα Pk για την επιλογή των λύσεων, η οποία εξαρτάται από το λόγο της απόστασης ΔC μιας λύσης από κάποιο τοπικό βέλτιστο προς μία μεταβλητή ελέγχου Θk , η οποία παραδοσιακά ονομάζεται θερμοκρασία. Η αρχική τιμή Θ_0 της θερμοκρασίας επιλέγεται υψηλή, έτσι ώστε η πιθανότητα ο αλγόριθμος να επιλέγει στα πρώτα του βήματα λύσεις, οι οποίες απέχουν από τα τοπικά βέλτιστα, να είναι μεγάλη. (Δημήτρης Παπαδάκης,1995)

Η αρχή λειτουργίας της προσομοιωμένης ανόπτησης είναι να ξεκινά από μία αρχική λύση και προχωρά σε κάθε επανάληψη σε μία καινούρια λύση στη γειτονιά της προηγούμενης λύσης μέχρι ένα κριτήριο τερματισμού να ικανοποιηθεί. Τα τρία βασικά κριτήρια τερματισμού που χρησιμοποιούνται είναι:

- Η βέλτιστη τιμή f δεν έχει βελτιωθεί για ένα ποσοστό τουλάχιστον κ1 συνεχών ανακυκλώσεων από Τ επαναλήψεις.
- Ο αριθμός των αποδεχούμενων κινήσεων είναι μικρότερος από ένα ποσοστό τουλάχιστον κ2 συνεχών ανακυκλώσεων από Τ επαναλήψεις.
- Έχει εκτελεστεί ένας μεγάλος αριθμός συνεχόμενων ανακυκλώσεων από Τ επαναλήψεις.

Παρακάτω παρατίθεται το διάγραμμα ροής της προσημειωμένης ανόπτησης:



Εικόνα 11 : Διάγραμμα ροής προσημειωμένης ανόπτησης

Κεφάλαιο 4

Υλοποίηση εφαρμογής

Στο παρόν κεφάλαιο περιγράφεται η δομή της εφαρμογής, καθώς και ο τρόπος λειτουργίας της. Οι τεχνολογίες που χρησιμοποιήθηκαν στην ανάπτυξη της εφαρμογής ήταν οι Java, Spring Framework, Maven, Optaplanner, Graphhopper, JPA, H2 database, Google Maps API, ενώ για την ανάπτυξη της εφαρμογής χρησιμοποιήθηκαν τα IDE Netbeans 8.2 και Eclipse Jee.

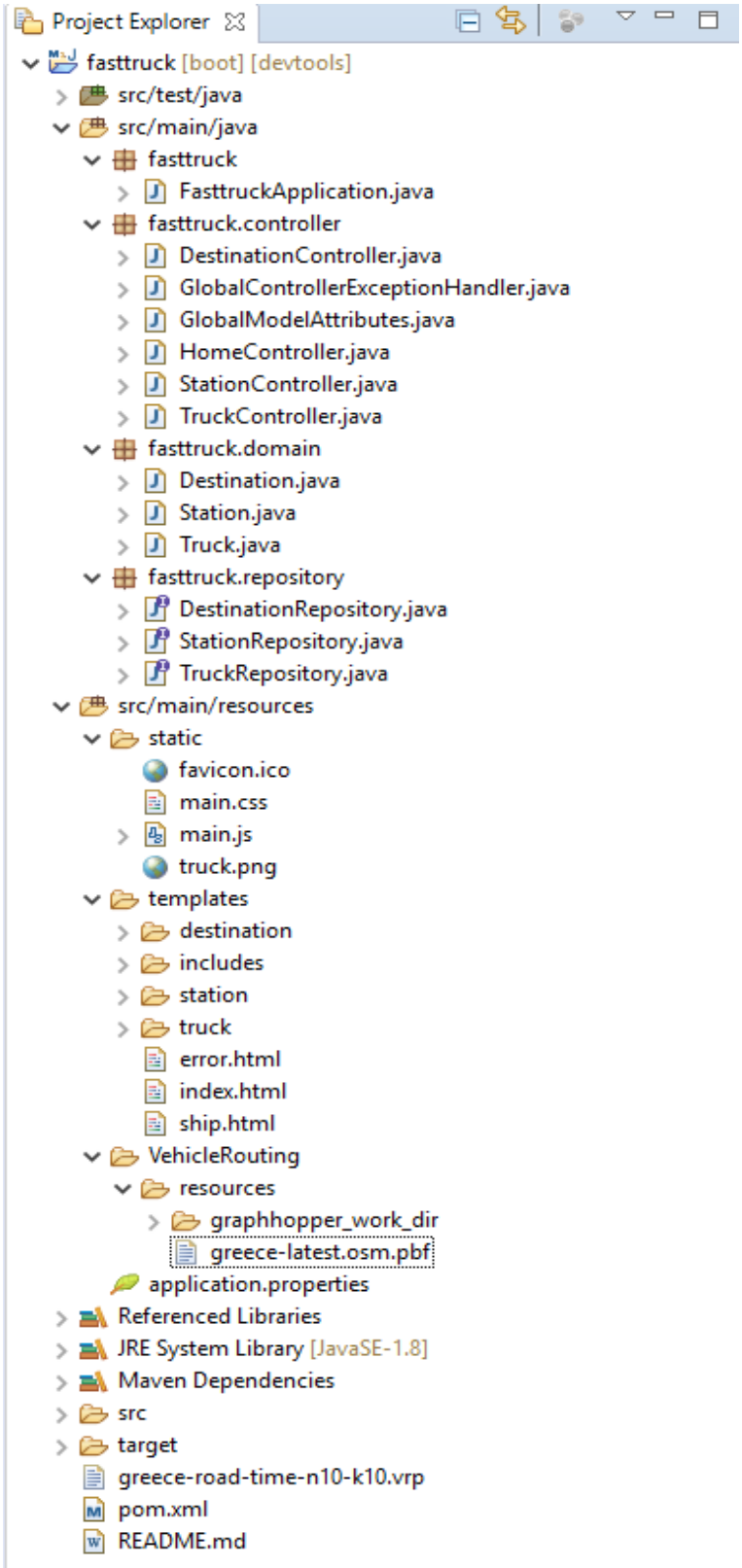
4.1 Σχεδίαση και υλοποίηση

Το Spring Framework αποτελεί ένα αξιόπιστο και σταθερό Framework, το οποίο επιτρέπει την εύκολη διαχείριση μίας διαδικτυακής εφαρμογής με μία βάση δεδομένων. Για αυτό το λόγο επιλέχτηκε για τη δημιουργία της εφαρμογής, που ονομάστηκε «fasttruck» στα πλαίσια της παρούσας εργασίας.

Για τη διαχείριση των εξαρτήσεων της εργασίας χρησιμοποιήθηκε το εργαλείο Maven με το οποίο ορίζονται οι σχέσεις κληρονομικότητας που έχει το project και οι βιβλιοθήκες που χρησιμοποιεί. Οι εξαρτήσεις αυτές δηλώνονται στο αρχείο «pom.xml», διατηρώντας τον υπόλοιπο κώδικα καθαρό. Την πρώτη φορά που ο προγραμματιστής κάνει build το project, τότε μόνο κατεβαίνουν από online πηγές οι βιβλιοθήκες που χρειάζεται το Project σε τοπικό φάκελο (συνήθως ο φάκελος .m2). Στη συνέχεια, κάθε φορά που γίνεται build το project, το maven ελέγχει αν υπάρχουν ήδη οι βιβλιοθήκες και ενημερώνει ότι είναι απαραίτητο.

Η δομή του project «fasttruck» φαίνεται στην Εικόνα 12. Στο αρχείο context.xml ορίζεται μόνο το μονοπάτι στο οποίο φορτώνεται το περιεχόμενο της εφαρμογής. Στο πακέτο «src/main/java» βρίσκεται ο κώδικας Java της εφαρμογής, ενώ στο πακέτο «/src/main/resources», άλλα αρχεία που αφορούν την εμφάνιση της εφαρμογής, όπως

html και εικόνες, καθώς και άλλα αρχεία που χρειάζονται από τις βιβλιοθήκες Optaplanner και Graphhoper.



Εικόνα 12: Δομή project "fasttruck".

Στο πακέτο «fasttruck» βρίσκεται το αρχείο App.java το οποίο εκκινεί την εφαρμογή. Στο πακέτο «fasttruck.controller» υπάρχουν τα αρχεία που είναι υπεύθυνα για τη λογική πίσω από κάθε σελίδα της εφαρμογής. Στο πακέτο «fasttruck.domain» είναι τα αρχεία που αντιστοιχούν στους πίνακες της βάσης δεδομένων. Στο πακέτο «fasttruck.repository» υπάρχουν interfaces με δήλωση μεθόδων που αφορούν ενέργειες πάνω στη βάση δεδομένων. Τα interfaces αυτά επεκτείνουν το υπάρχον interface «CrudRepository» που διαθέτει το Spring Framework. Με αυτό τον τρόπο πετυχαίνουμε να γράψουμε λιγοστές γραμμές κώδικα για τη διαχείριση της βάσης δεδομένων.

4.2 Βάση Δεδομένων

Στην ενότητα αυτή παρουσιάζεται η βάση δεδομένων που υλοποιήθηκε και αποτελείται από τρεις πίνακες. Τον πίνακα Station που αποθηκεύει τα δεδομένα που αφορούν τους σταθμούς, τον πίνακα Truck όπου αποθηκεύονται τα δεδομένα που αφορούν τα φορτηγά που υπάρχουν σε κάθε σταθμό και τον πίνακα Destination που περιέχει τα δεδομένα που αφορούν τους προορισμούς των φορτηγών.

Στον Πίνακας 2 παραθέτουμε τον πίνακα Station της βάσης δεδομένων με τα πεδία και τον τύπο τους. Ο τύπος των δεδομένων ορίζεται στη Java και το JPA αναλαμβάνει την αντιστοίχιση με τους τύπους που θα αποθηκευτούν τελικά στη βάση. Κλειδί του πίνακα είναι το id. Στον πίνακα αποθηκεύουμε το όνομα και τη διεύθυνση του σταθμού, καθώς και τις γεωγραφικές συντεταγμένες του. Τέλος, αποθηκεύουμε μία λίστα με τα φορτηγά (Trucks) που διαθέτει. Ένας σταθμός μπορεί να διαθέτει πολλά φορτηγά και η σχέση αυτή απεικονίζεται και για τη βάση δεδομένων με το annotation «@OneToMany(mappedBy = "station")» για το JPA.

Πεδίο	Τύπος
id (KEY)	Long
name	String
address	String
latitude	Double
longitude	Double
trucks	List<Truck>

Πίνακας 2: Πίνακας "Station" της Βάσης Δεδομένων.

Στον Πίνακας 3 παραθέτουμε τον πίνακα Truck της βάσης δεδομένων με τα πεδία και τον τύπο τους. Κλειδί του πίνακα είναι το id. Στον πίνακα αποθηκεύουμε μία ονομασία για το φορτηγό, καθώς και τη χωρητικότητά του. Τέλος, αποθηκεύουμε το σταθμό στον οποίο ανήκει το φορτηγό. Πολλά φορτηγά μπορούν να ανήκουν σε ένα μόνο σταθμό και η σχέση αυτή απεικονίζεται και για τη βάση δεδομένων με το annotation «@ManyToOne(optional=false)» για το JPA.

Πεδίο	Τύπος
id (KEY)	Long
name	String
volume	Double
station	Station

Πίνακας 3: Πίνακας "Truck" της Βάσης Δεδομένων.

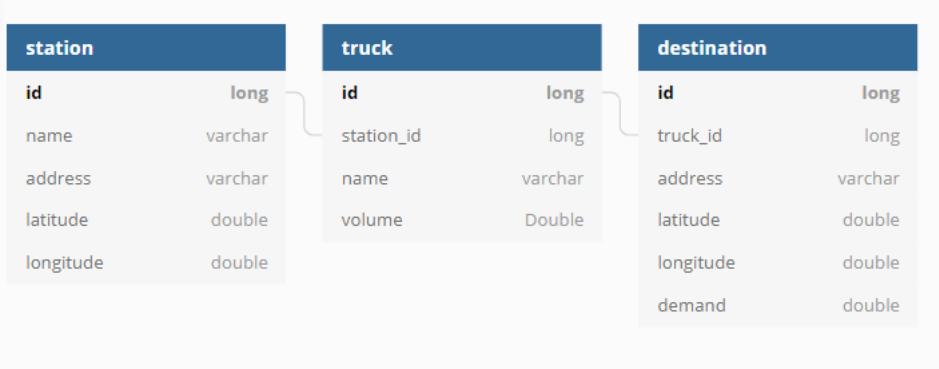
Στον Πίνακας 4 παραθέτουμε τον πίνακα Destination της βάσης δεδομένων με τα πεδία και τον τύπο τους. Κλειδί του πίνακα είναι το id. Στον πίνακα αποθηκεύουμε τη διεύθυνση του προορισμού, καθώς και τις γεωγραφικές συντεταγμένες του. Τέλος, αποθηκεύουμε τις απαιτήσεις που έχει ο πελάτης του προορισμού. Οι απαιτήσεις

αντιστοιχούν στη χωρητικότητα του φορτηγού που καταλαμβάνει ο συγκεκριμένος πελάτης.

Πεδίο	**Τύπος**
id (KEY)	Long
address	String
latitude	Double
longitude	Double
demand	Double

Πίνακας 4: Πίνακας "Destination" της Βάσης Δεδομένων.

Στην Εικόνα 13 φαίνεται το ER διάγραμμα των πινάκων της βάσης δεδομένων στην παραδοσιακή μορφή όπου εμφανίζονται οι συσχετίσεις των πινάκων. Για να φανούν οι σχέσεις προσθέσαμε στον πίνακα Truck το πεδίο station_id το οποίο δείχνει στο κλειδί του πίνακα station και το πεδίο truck_id στον πίνακα destination, το οποίο δείχνει στο κλειδί του πίνακα truck. Με αυτόν τον τρόπο δηλώνεται ότι κάθε σταθμός μπορεί να διαθέτει πολλά φορτηγά και κάθε φορτηγό μπορεί να επισκεφτεί πολλούς προορισμούς.



Εικόνα 13: Διάγραμμα ER.

Στον Πίνακας 5: Κώδικας «Station.java». παραθέτουμε τον πηγαίο κώδικα του αρχείου «Station.java», το οποίο ανήκει στο πακέτο fasttruck.domain και καθορίζει το mapping που γίνεται για τη βάση δεδομένων μέσω του JPA, το οποίο εισάγεται με

την εντολή «*import javax.persistence.*;*». Το annotation *@Entity* καθορίζει ότι η κλάση Station που ακολουθεί, αντιστοιχίζεται με έναν πίνακα στη βάση δεδομένων και οι ιδιότητές της που ακολουθούνται από το annotation *@Column* αντιστοιχούν στις στήλες του πίνακα. Στα annotations του JPA υπάρχει η δυνατότητα για επιπλέον παραμετροποίηση των στηλών του πίνακα. Για παράδειγμα δεν επιτρέπουμε σε καμία στήλη του πίνακα να περιέχει null τιμή, θέτοντας την τιμή false στο attribute nullable. Επιπλέον, στην στήλη name που αντιστοιχεί στο όνομα του σταθμού, θέσαμε το attribute unique στην τιμή true, υποχρεώνοντας το όνομα του κάθε σταθμού να είναι μοναδικό. Το κλειδί του πίνακα χαρακτηρίζεται από το annotation *@Id*, ενώ μπορούμε να ορίσουμε με το annotation *@GeneratedValue* τον τρόπο δημιουργίας του (επιλέχτηκε η αυτόματη δημιουργία του από το JPA για κάθε εγγραφή). Επιπροσθέτως, η σχέση που περιεγράφηκε παραπάνω με τον κάθε σταθμό να διαθέτει πολλά φορτηγά, απεικονίζεται με το annotation *@OneToMany* πριν από τη λίστα με τα φορτηγά.

Ακολουθεί ο κατασκευαστής της κλάσης που δημιουργεί ένα αντικείμενο, καθώς και οι setters και getters των ιδιοτήτων της κλάσης. Ο τύπος Truck οδηγεί στο αρχείο Truck.java που είναι γραμμένο με την ίδια λογική και η υλοποίηση του αντιστοιχεί στον Πίνακας 3. Με τον ίδιο τρόπο έχει γραφτεί και το αρχείο Destination για τον προορισμό που περιεγράφηκε στον Πίνακας 4.

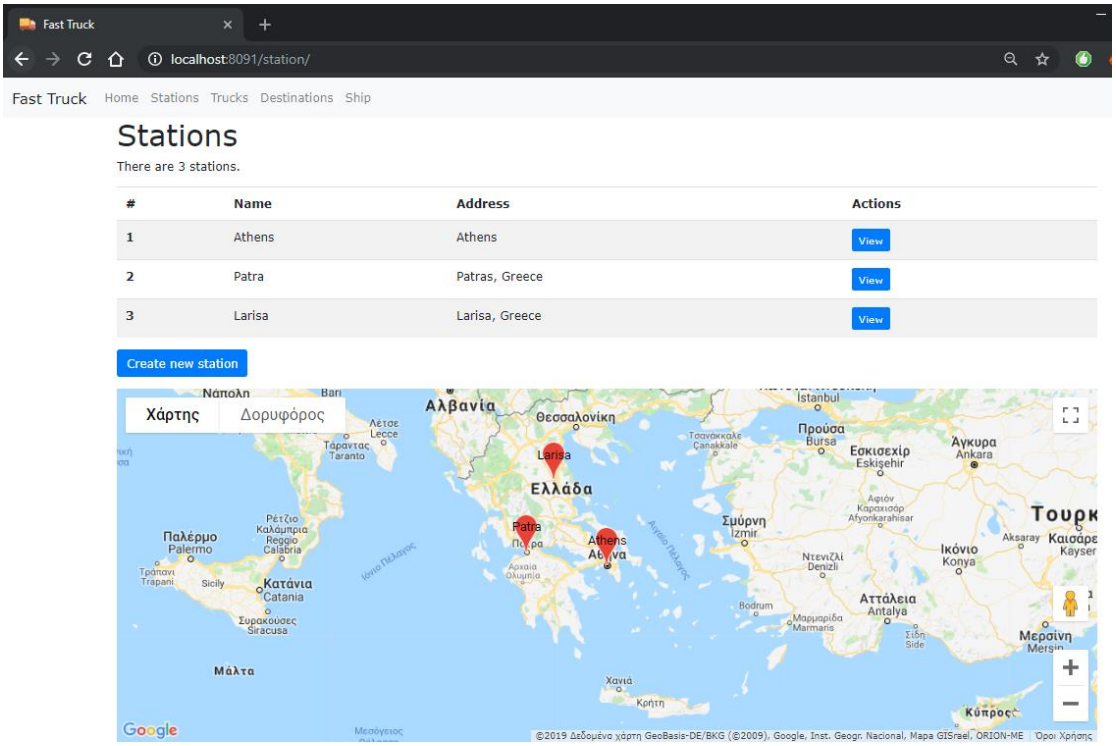
```
package fasttruck.domain;

import javax.persistence.*;
import java.io.Serializable;
import java.util.List;

@Entity
public class Station implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Column(nullable = false, unique = true)
    private String name;
```

```
    @Column(nullable = false)
    private String address;

    @Column(nullable = false)
    private Double latitude;

    @Column(nullable = false)
    private Double longitude;

    @OneToMany(mappedBy = "station")
    private List<Truck> trucks;

    public Station() {
        id = 0L;
        name = "";
        address = "";
        latitude = 0.0;
        longitude = 0.0;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }
```

```
    public Double getLatitude() {
        return latitude;
    }

    public void setLatitude(Double latitude) {
        this.latitude = latitude;
    }

    public Double getLongitude() {
        return longitude;
    }

    public void setLongitude(Double longitude) {
        this.longitude = longitude;
    }

    public List<Truck> getTrucks() {
        return trucks;
    }
}
```

Πίνακας 5: Κώδικας «Station.java».

Η εμφάνιση της εφαρμογής της Εικόνα 14 καθορίζεται από τον κώδικα του αρχείου «index.html» που παρατίθεται στον Πίνακας 6 και ανήκει στο πακέτο «templates/station».



Εικόνα 14: Σελίδα Stations.

Η πρώτη κι η τελευταία γραμμή του αρχείου μας επιτρέπει να εισάγουμε το header και το footer που είναι κοινά για όλες τις σελίδες. Στη συνέχεια ανάμεσα στα tags <table> </table> ορίζουμε τον πίνακα στον οποίο παραθέτουμε τις εγγραφές των σταθμών. Με τα διπλά άγκιστρα καθορίζουμε ότι το περιεχόμενο τους θα αντικατασταθεί στο runtime από την τιμή μίας μεταβλητής, π.χ. η θέση {{name}}) θα αντικατασταθεί από το όνομα κάθε φορτηγού. Επίσης, ορίζουμε επαναληπτικά γραμμές του πίνακα περικλείοντας τον κώδικα που αντιστοιχεί σε μία γραμμή στα tags {{#stations}} {{stations}}. Τέλος ο κώδικας JavaScript που περικλείεται στα tags <script> </script> μας δίνει τη δυνατότητα να εμφανίσουμε τους σταθμούς στους χάρτες της Google.

```
{{>includes/header}}

<h1>Stations</h1>
<p>There are {{stationCount}} stations.</p>

<table class="table table-striped">
    <thead>
    <tr>
        <th scope="col">#</th>
        <th scope="col">Name</th>
        <th scope="col">Address</th>
        <th scope="col">Actions</th>
    </tr>
    </thead>
    <tbody>
    {{#stations}}
    <tr>
        <th scope="row">{{id}}</th>
        <td>{{name}}</td>
        <td>{{address}}</td>
        <td>
            <a     class="btn     btn-primary     btn-sm"     href="/station/{{id}}"
role="button">View</a>
        </td>
    </tr>
    {{/stations}}
    </tbody>
</table>

<a   class="btn   btn-primary"   href="/station/create"   role="button">Create   new
station</a>

<div id="map" class="my-3"></div>

<script>
function initMap() {
    var map = new google.maps.Map(document.getElementById('map'), {
        zoom: {{googleMapsZoom}},
        center: { lat: 37.9838, lng: 23.7275 } // Athens
    });
```

```
    {{#stations}}
    new google.maps.Marker({
        position: {lat: {{latitude}}, lng: {{longitude}}},
        map: map,
        label: '{{name}}'
    });
    {{/stations}}
}
</script>
<script async defer

src="https://maps.googleapis.com/maps/api/js?key={{googleMapsApiKey}}&callback=init
Map">
</script>

{{>includes/footer}}
```

Πίνακας 6: Κώδικας «index.html» για Stations.

Ο κώδικας που ελέγχει το περιεχόμενο της σελίδας της Εικόνα 14: Σελίδα Stations. παρουσιάζεται στον Πίνακας 7. Η κλάση χαρακτηρίζεται από το annotation *@Controller* το οποίο δηλώνει ότι πρόκειται για μία κλάση που διαχειρίζεται μία ιστοσελίδα. Το annotation *@GetMapping("/station")* δηλώνει πως όταν ο χρήστης επισκεφτεί τη σελίδα αυτή, τότε θα κληθεί η μέθοδος αυτή η οποία τελικά επιστρέφει τη σελίδα index που παρουσιάσαμε στον προηγούμενο πίνακα. Η μέθοδος αυτή θέτει τιμές στα μοντέλα της σελίδας. Αρχικά καλεί τη μέθοδο *count()* του *stationRepository*, που θα επιστρέψει πόσοι σταθμοί υπάρχουν συνολικά στη βάση. Έπειτα καλεί τη μέθοδο *findAll()* του *stationRepository* ώστε να λάβει σε μία λίστα όλους τους σταθμούς που υπάρχουν.

Οι δύο αυτοί μέθοδοι υπάρχουν στην κλάση *CrudRepository* του Spring Framework, η οποία γίνεται extend από την κλάση *StationRepository*. Οι τιμές που επιστρέφονται από τις μεθόδους αυτές, τίθενται ως attributes στο μοντέλο της σελίδας. Για κάθε σελίδα στην οποία μπορεί να οδηγηθεί ο χρήστης από τη σελίδα των σταθμών (παρουσιάζονται στη συνέχεια), δημιουργήθηκε η αντίστοιχη μέθοδος στην κλάση αυτή, καθώς και το αντίστοιχο HTML αρχείο.

```
@Controller
public class StationController {
    @Autowired
    private StationRepository stationRepository;

    /**
     * GET /station
     * Lists all stations.
     */
    @GetMapping("/station/")
    public String index(Model model) {
        long stationCount = stationRepository.count();
        model.addAttribute("stationCount", stationCount);
        Iterable<Station> stations = stationRepository.findAll();
        model.addAttribute("stations", stations);
        return "station/index";
    }
}
```

Πίνακας 7: Τμήμα από «StationController.java»

4.3 Επίλυση προβλήματος

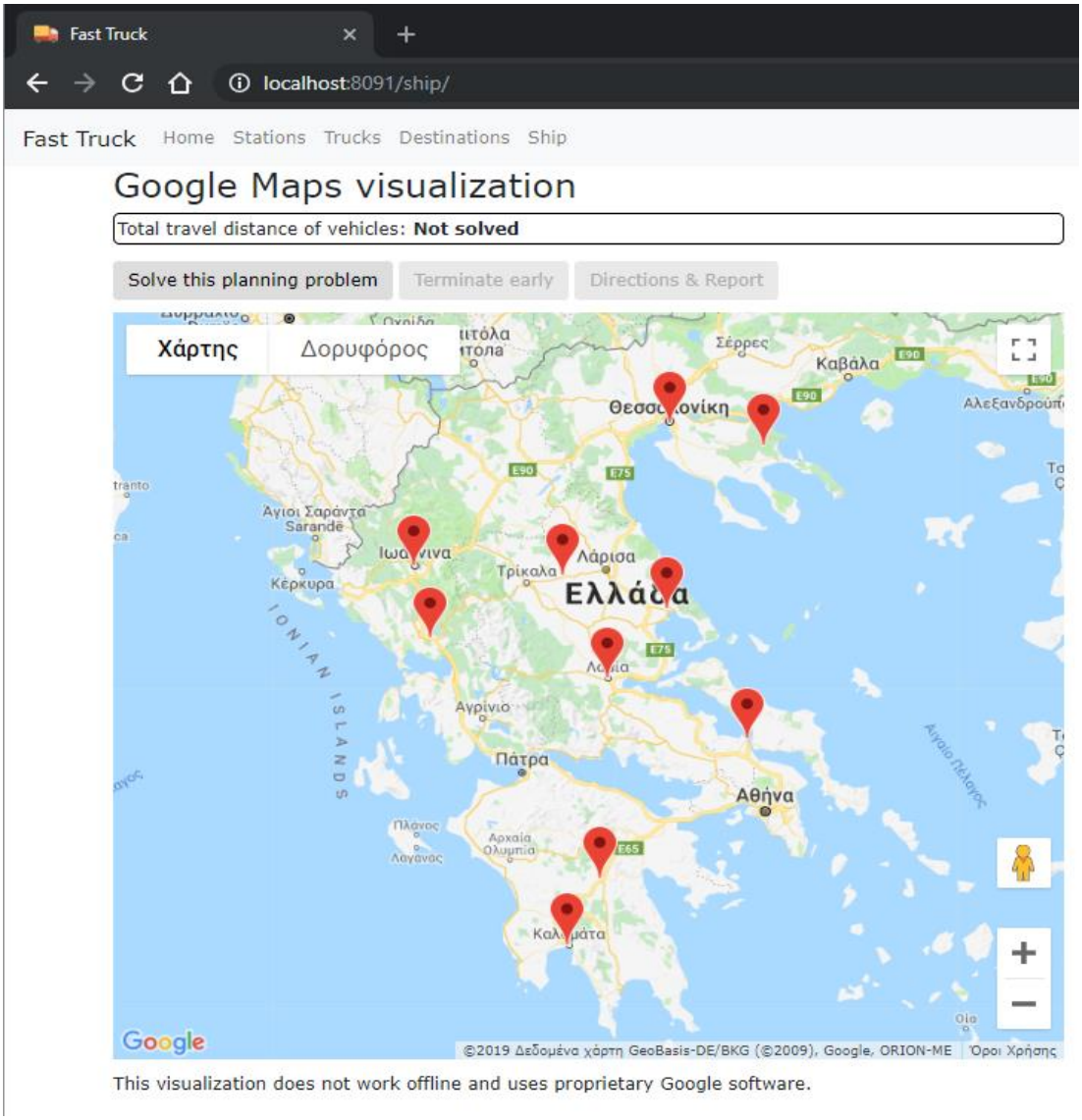
Μέχρι το σημείο αυτό έχουμε παρουσιάσει το πώς ορίζουμε το πρόβλημα και πως το αποθηκεύουμε και το διαχειριζόμαστε σε σχέση με τη βάση δεδομένων. Για να επιλυθεί το πρόβλημα επιλέχτηκε η βιβλιοθήκη Optaplanner, καθώς και το GraphHopper Directions API. Το GraphHopper υπολογίζει όλες τις αποστάσεις μεταξύ των προορισμών σε χιλιόμετρα ακολουθώντας τις πραγματικές αποστάσεις των δρόμων, δεδομένα που χρειάζεται το Optaplanner για να επιλύσει το πρόβλημα.

Λόγω του διαφορετικού τρόπου διαχείρισης των αρχείων jar και war από το Spring και το Optaplanner, επιλέχτηκε να τρέχει παράλληλα τοπικά το project Optaplanner-webexamples που παρέχεται στη ιστοσελίδα της βιβλιοθήκης σε διαφορετικό port από το project fasttruck. Οι εφαρμογές έγιναν τοπικά deploy στον server wildfly, που προτείνεται από τους δημιουργούς της βιβλιοθήκης Optaplanner, με το fasttruck να χρησιμοποιεί το Port 8091 και το Optaplanner-webexamples να χρησιμοποιεί το Port 8080. Το fasttruck αποκτά πρόσβαση στα services που είναι διαθέσιμα τοπικά στο Port 8080. Για παράδειγμα όταν ο χρήστης πατήσει το κουμπί «Solve this planning problem» της εφαρμογής fasttruck, θα κληθεί η συνάρτηση που παρουσιάζεται στον Πίνακας 8. Η συγκεκριμένη μέθοδος αυτή καλεί το service που παρέχεται στο link http://localhost:8080/optaplanner-webexamples/rest/vehiclerouting/solution/solve. Με παρόμοιο τρόπο καλούνται και τα υπόλοιπα services της Optaplanner.

```
  solve = function() {
    $('#solveButton').attr("disabled", "disabled");
    $('#resolveDirectionsButton').attr("disabled", "disabled");
    if (directionsTaskTimer != undefined) {
      window.clearInterval(directionsTaskTimer);
      directionsTaskTimer = undefined;
    }
    $.ajax({
      url:                                          "http://localhost:8080/optaplanner-
webexamples/rest/vehiclerouting/solution/solve",
      type: "POST",
      dataType : "json",
      data : "",
      success: function(message) {
        console.log(message.text);
        intervalTimer = setInterval(function () {
          updateSolution()
        }, 2000);
        $('#terminateEarlyButton').removeAttr("disabled");
      },  error  :  function(jqXHR,  textStatus,  errorThrown)  {ajaxError(jqXHR,
textStatus, errorThrown)}
    });
  };
```

Πίνακας 8: Μέθοδος solve του fasttruck.

Όταν ο χρήστης οριστικοποιήσει τις παραμέτρους του προβλήματος, πατάει το κουμπί Ship για την επίλυση του προβλήματος. Πριν κληθεί η βιβλιοθήκη Optaplanner πρέπει να υπολογιστούν οι αποστάσεις μεταξύ των προορισμών. Για να γίνει αυτό, τοποθετήθηκε το αρχείο http://download.geofabrik.de/europe/greece-latest.osm.pbf, το οποίο περιέχει τις πληροφορίες για το οδικό δίκτυο της Ελλάδας στο φάκελο /src/main/resources/VehicleRouting.resources. Την πρώτη φορά που θα επισκεφτεί ο χρήστης τη σελίδα αυτή, η εφαρμογή θα κάνει κάποια δευτερόλεπτα να φορτώσει, καθώς υπολογίζονται οι αποστάσεις μέσω του GraphHopper API. Όσο εκτελείται ο υπολογισμός αυτός, γράφονται δεδομένα στο φάκελο /src/main/resources/VehicleRouting.resources.graphhopper_work_dir. Την επόμενη φορά που θα ακολουθηθεί ο σύνδεσμος Ship, η μεταφορά θα είναι πιο άμεση.



Εικόνα 15: Σελίδα Ship.

Υπεύθυνο για τη διαδικασία που αναφέρθηκε παραπάνω με το GraphHopper είναι το αρχείο HomeController.java. Αφού ολοκληρωθεί η διεργασία αυτή, το HomeController αναλαμβάνει να γράψει όλες τις απαιτούμενες για τη βιβλιοθήκη Optaplanner πληροφορίες σε ένα αρχείο .vrp. Το αρχείο αυτό περιέχει τα απαραίτητα στοιχεία για τους σταθμούς, για τους προορισμούς και τις παραμέτρους του προβλήματος.

Το αρχείο που παράχθηκε για το συγκεκριμένο παράδειγμα παρουσιάζεται στον Πίνακας 9. Η διάσταση του προβλήματος για το παράδειγμα είναι 13, που συντίθεται από το πλήθος των σταθμών (3) και το πλήθος των προορισμών (10). Θέτοντας την παράμετρο EDGE_WEIGHT_TYPE=EXPLICIT ορίζουμε στον Optaplanner να υπολογίσει τη βέλτιστη διαδρομή χρησιμοποιώντας τις ακριβείς τιμές που δίνονται στην ενότητα EDGE_WEIGHT_SECTION. Στο σημείο εκείνο ορίζεται ένας δισδιάστατος πίνακας όπου καταγράφονται οι αποστάσεις μεταξύ κάθε κόμβου του γράφου (σταθμοί και προορισμοί) εκφρασμένος σε χιλιόμετρα, όπως ορίζουμε και στην παράμετρο EDGE_WEIGHT_UNIT_OF_MEASUREMENT. Οι τιμές αυτές έχουν προκύψει από τον υπολογισμό που έκανε νωρίτερα το GraphHopper. Επιπλέον στην ενότητα NODE_COORD_SECTION καταγράφονται οι γεωγραφικές συντεταγμένες κάθε κόμβου, ενώ στην ενότητα DEPOT_SECTION καταγράφονται τα ID των κόμβων που αντιστοιχούν στους σταθμούς.

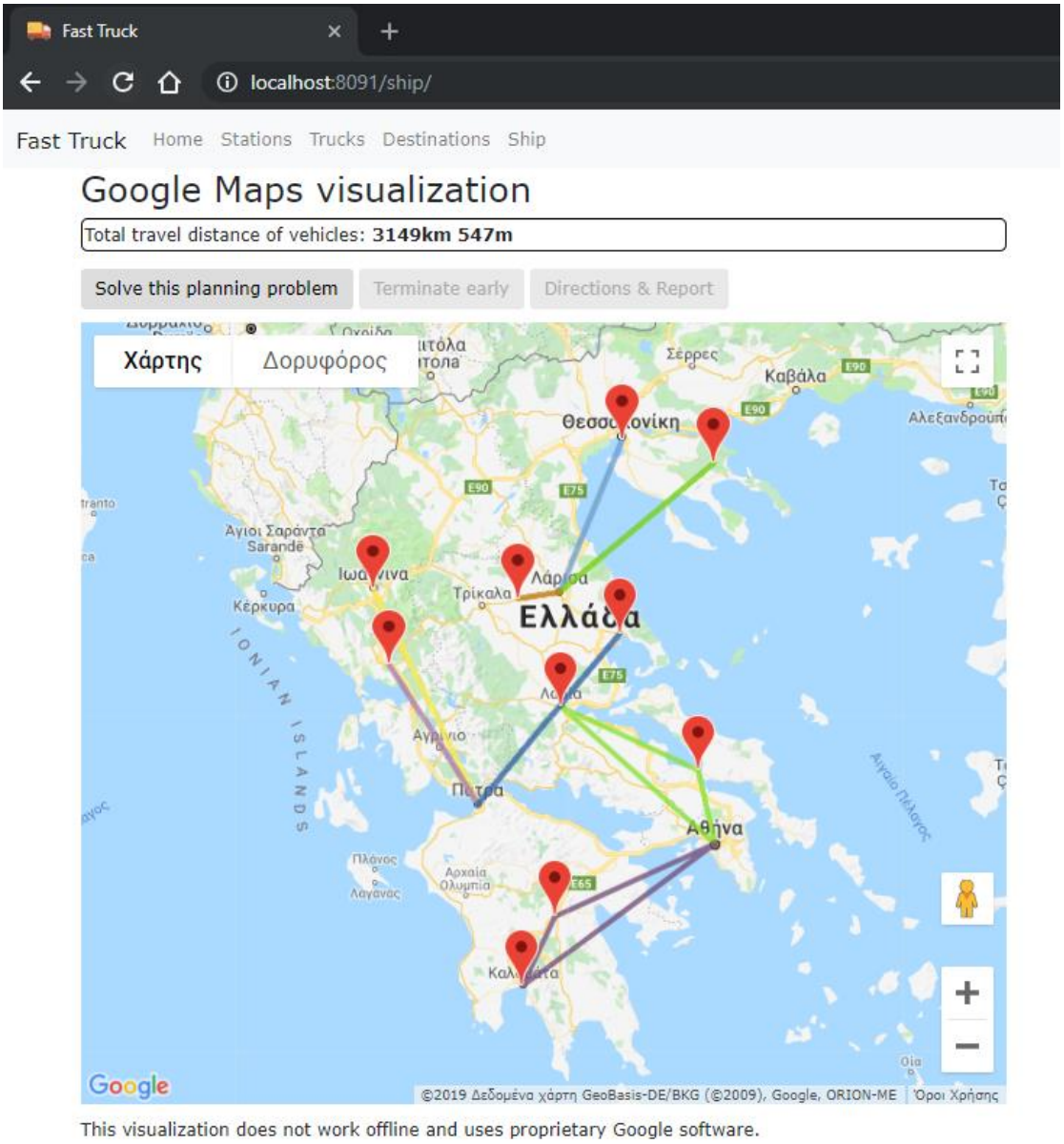
```
NAME: Greece
COMMENT: Generated for FastTruck.
TYPE: CVRP
DIMENSION: 13
EDGE_WEIGHT_TYPE: EXPLICIT
EDGE_WEIGHT_FORMAT: FULL_MATRIX
EDGE_WEIGHT_UNIT_OF_MEASUREMENT: km
CAPACITY: 400
NODE_COORD_SECTION
1 37.9838096 23.7275388 Station1
2 38.2466395 21.734574 Station2
3 39.63902239999999 22.4191254 Station3
1034 40.61880439999999 22.9552561 Destination1034
1035 37.504974 22.375043 Destination1035
1036 38.8898017 22.4290785 Destination1036
1037 38.47255639999999 23.5864426 Destination1037
1038 39.59561739999999 22.0680089 Destination1038
```

```
1039 37.0350132 22.1087589 Destination1039
1040 40.4703942 23.7156809 Destination1040
1041 39.1623421 20.9878775 Destination1041
1042 39.658784 20.8510925 Destination1042
1033 39.3679537 22.9308555 Destination1033
EDGE_WEIGHT_SECTION
0.0   210.71419158534272   351.7161421885417   501.2187881664969   157.6326293266408
211.3675027188333     76.16254605261072     324.0652712212551     237.04318870644414
593.1403408361616 346.8212981696637 414.578864508626 323.2985616796754
210.93731475063    0.0    320.45724434984015    468.6505313277945    203.097861487939
180.10860488013154     276.9485342139092     360.8539543825539     282.5084208677425
560.5720839974599 153.10341133096173 220.86097766992373 292.03966384097384
351.62312238248   320.41451737849076   0.0   153.9542629596456   455.14965311979023
149.11216851198188     300.2611558457593     35.26036801440424     534.5602124995934
245.87581562931027 247.05450596281221 186.7514383017741 58.57358347282406
501.1811358404464   468.6995568364573   154.7415024396569   0.0   604.7076665777562
298.67018196994815     449.81916930372563     188.73830047237064     684.1182259575598
92.10935408727684 322.73916442077814 262.4360967597404 208.1315969307906
157.7485218181249   202.41796881413572   455.8902674173347   605.3929133952914   0.0
315.54162794762647     223.7597412814038     428.2393964500485     81.27980993523732
697.3144660649562 338.5250753984567 406.282641737419 427.4726869084684
210.7451132103174     179.53650820632808     149.2330828095277     298.7357287874825
314.2716439476268   0.0   159.38314667359666   110.6614618422416   393.6822033274301
390.6572814571482 207.0664447906494 244.37852712961126 120.8155023006614
75.50825754409475     249.42319154010576     305.8368271433049     455.33947312126014
196.341629281404   165.48818767359643   0.0   278.18595617601864   275.75218866120736
547.2610257909262 385.53029812442685 453.28786446338916 277.41924663443865
323.57022571273956     360.4351867087508     35.76646131195008     189.72072328990546
427.0967564500493   110.73455284224154   272.208259176019   0.0   506.50731582985264
281.64227595957027 214.47479429307202 154.17172663203388 91.97452880308379
237.41257929632087     282.0820262923317     535.5543248955315     685.056970873488
82.71937803363001   395.2056854258232   303.42379875959966   507.90345392824446   0.0
776.9785235431527 418.18913287665305 485.94669921561535 507.13674438666516
615.7429673276472     583.2613883236569     269.30333392685657     91.77783790481162
719.2694980649543     413.2320134571478     564.3810007909257     303.3001319595702
798.6800574447581 0.0 437.3009959079775 376.9979282469399 322.69342841799016
346.65492329727147     152.3876662932827     246.70794089648206     322.5150788744373
338.81547003458104     207.29001342677356     412.66614276055094     214.71850192919598
418.2260294143839 414.4366315441029 0.0 74.72552521656567 302.9160083876155
415.2073416588998     220.94008465491083     186.9021932581102     262.7093312360657
407.3678883962094     243.9904567884016     481.21856112217927     154.91275429082413
486.77844777601223 354.63088390573125 74.97969223923195 0.0 243.1102607492439
323.9375611711595     292.7289561671704     58.08426677036998     207.58691274832523
427.4640919084693     121.42660730066132     272.5755946344389     91.40900180308378
```

```
506.87465128827256 299.5084654179901 303.20313975149156 242.9000720904536 0.0
DEMAND_SECTION
1 0
2 0
3 0
1034 350
1035 320
1036 190
1037 140
1038 270
1039 80
1040 380
1041 340
1042 310
1033 220
DEPOT_SECTION
1
2
3
-1
EOF
```

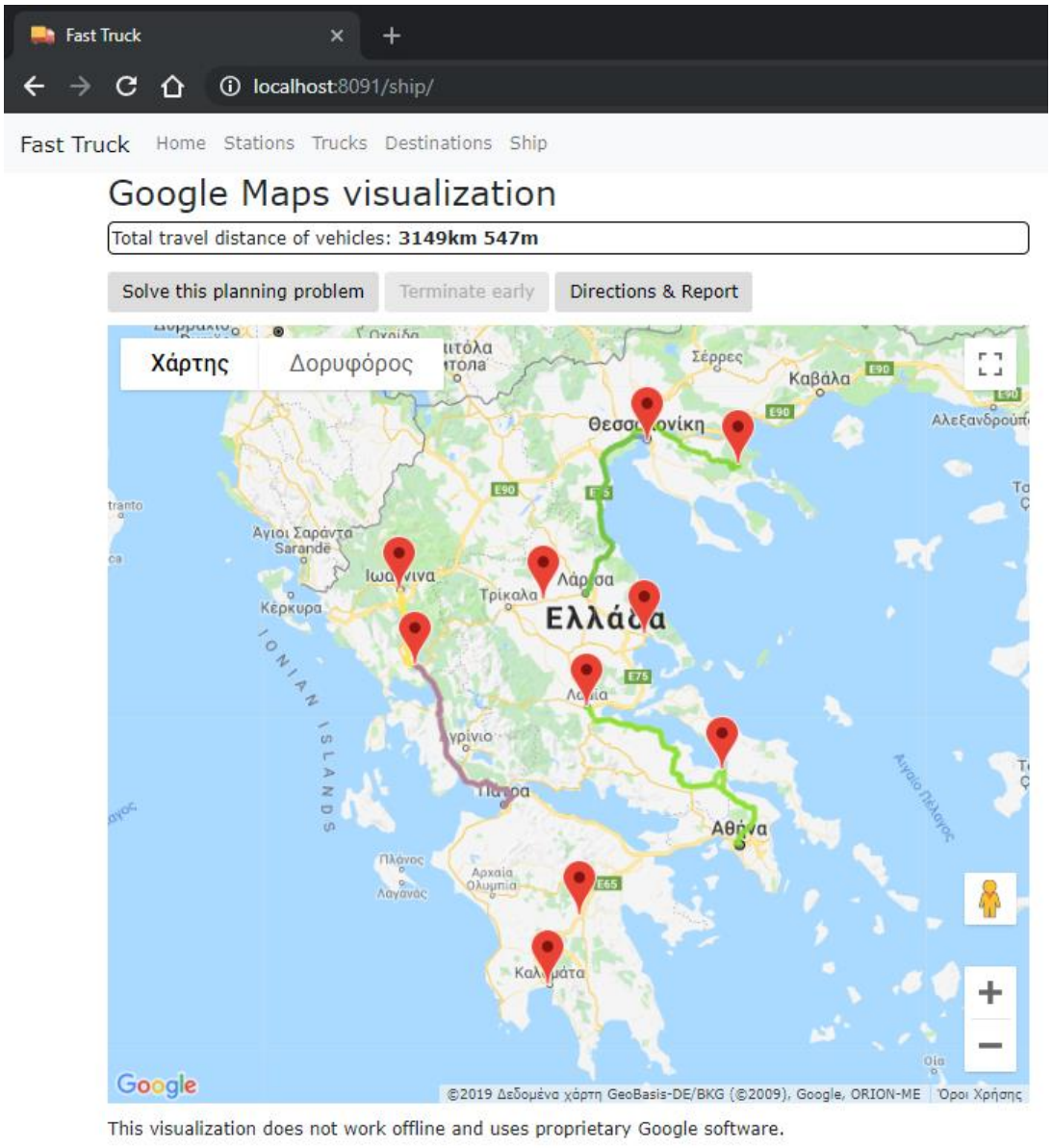
Πίνακας 9: Αρχείο vrp.

Όταν έχει φορτωθεί η σελίδα του Ship, ο χρήστης βλέπει στο χάρτη τους προορισμούς που έχουν οριστεί νωρίτερα και αν πατήσει το κουμπί «Solve this planning problem» θα καλέσει το service της βιβλιοθήκης Optaplanner για την επίλυση του προβλήματος. Όταν ο αλγόριθμος επιλύσει το πρόβλημα, τότε εμφανίζεται η σελίδα που παρουσιάζεται στην Εικόνα 16 και δείχνει τις διαδρομές σε ευθείες γραμμές από κάθε σταθμό προς τους προορισμούς. Ο χρόνος επίλυσης αυξάνεται με το μέγεθος του προβλήματος. Το ίδιο χρώμα γραμμής αντιστοιχεί στο ίδιο φορτηγό. Για παράδειγμα, από την Αθήνα με βάση τις ανάγκες του προβλήματος χρειάστηκαν μόνο δύο φορτηγά για τη μεταφορά των αγαθών. Επίσης, πάνω από το χάρτη εμφανίζεται η συνολική απόσταση που πρέπει να διανύσουν τα φορτηγά.



Εικόνα 16: Λύση του προβλήματος.

Αν ο χρήστης πατήσει το κουμπί Terminate early ενεργοποιείται και το κουμπί Directions, το οποίο αν το πατήσει ο χρήστης, τότε εμφανίζονται σταδιακά οι πραγματικές διαδρομές που θα εκτελέσουν τα φορτηγά, όπως φαίνεται στην Εικόνα 17: Απεικόνιση αποτελεσμάτων.

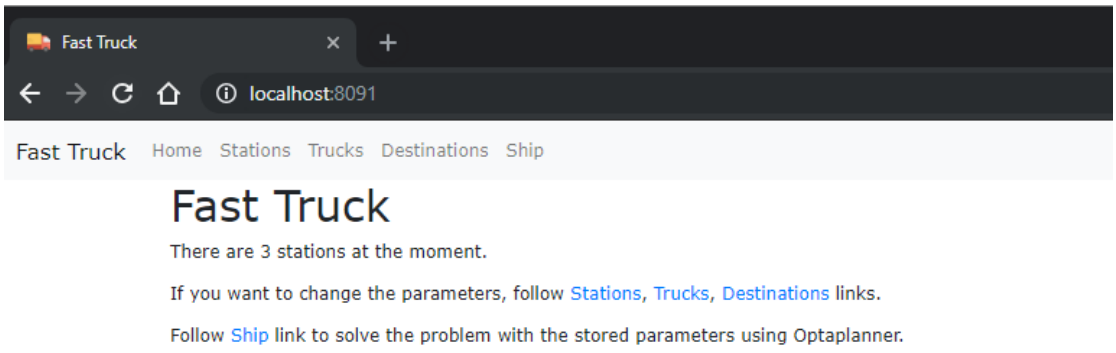


Εικόνα 17: Απεικόνιση αποτελεσμάτων

Κεφάλαιο 5

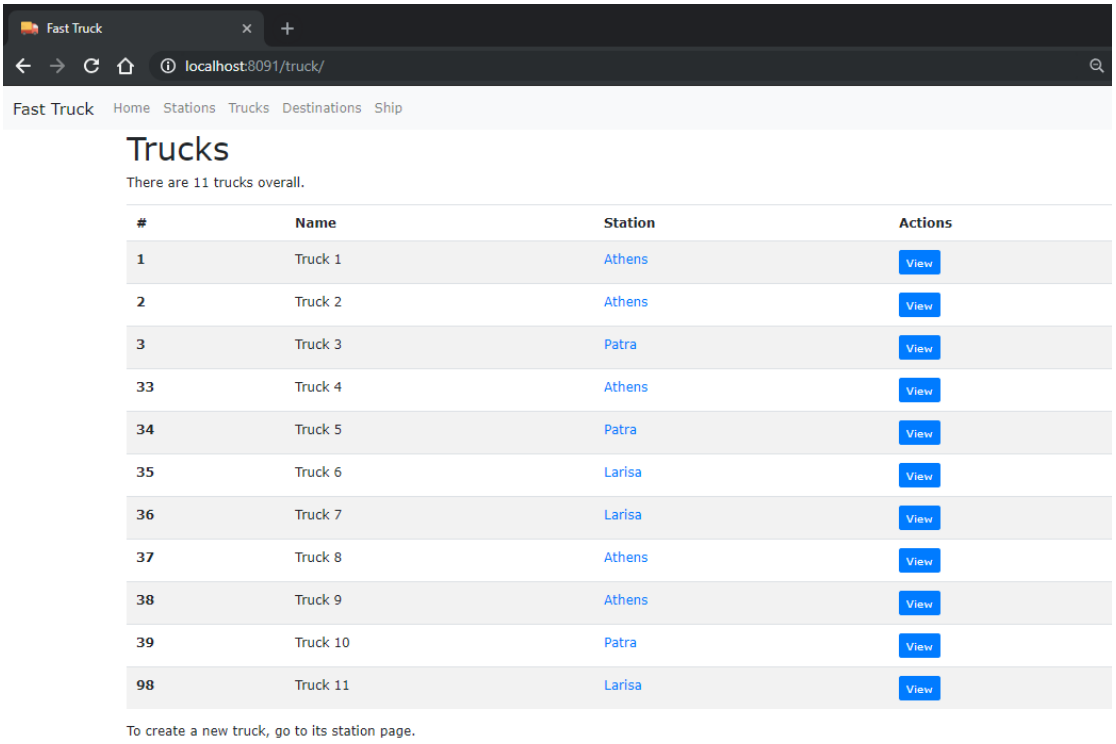
Διαδικτυακή Εφαρμογή

Στην ενότητα αυτή παρουσιάζουμε το Web User Interface της εφαρμογής μας. Η αρχική σελίδα της εφαρμογής «fasttruck» παρουσιάζεται στην Εικόνα 18.Εικόνα 18: Αρχική σελίδα fasttruck. Στη σελίδα αυτή, για την οποία υπεύθυνο είναι το αρχείο «HomeController.java» παρουσιάζονται κάποιες γενικές πληροφορίες για την εφαρμογή. Στους συνδέσμους Stations, Trucks, Destinations μπορούν να τροποποιηθούν οι παράμετροι του προβλήματος, ενώ αν πατήσουμε το σύνδεσμο Ship, τότε ξεκινάει η διαδικασία επίλυσης του προβλήματος, την οποία θα περιγράψουμε στην επόμενη ενότητα.



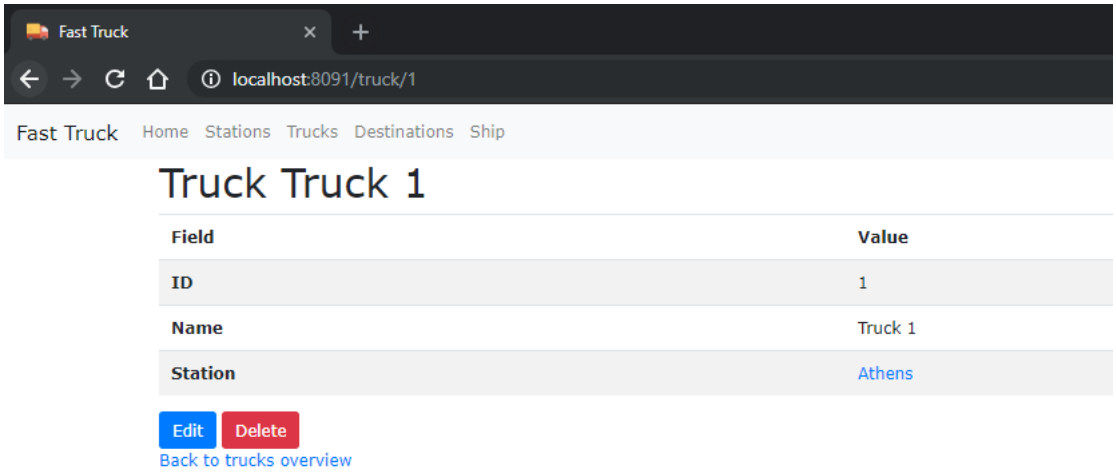
Εικόνα 18: Αρχική σελίδα fasttruck.

Αν ακολουθήσουμε το σύνδεσμο Trucks βλέπουμε τις πληροφορίες για τα φορτηγά που υπάρχουν στο σύστημα, όπως φαίνεται στην Εικόνα 19. Αν θέλουμε να προσθέσουμε ένα φορτηγό, ακολουθούμε το σύνδεσμο για να πάμε στο σταθμό που επιθυμούμε να καταχωρηθεί.



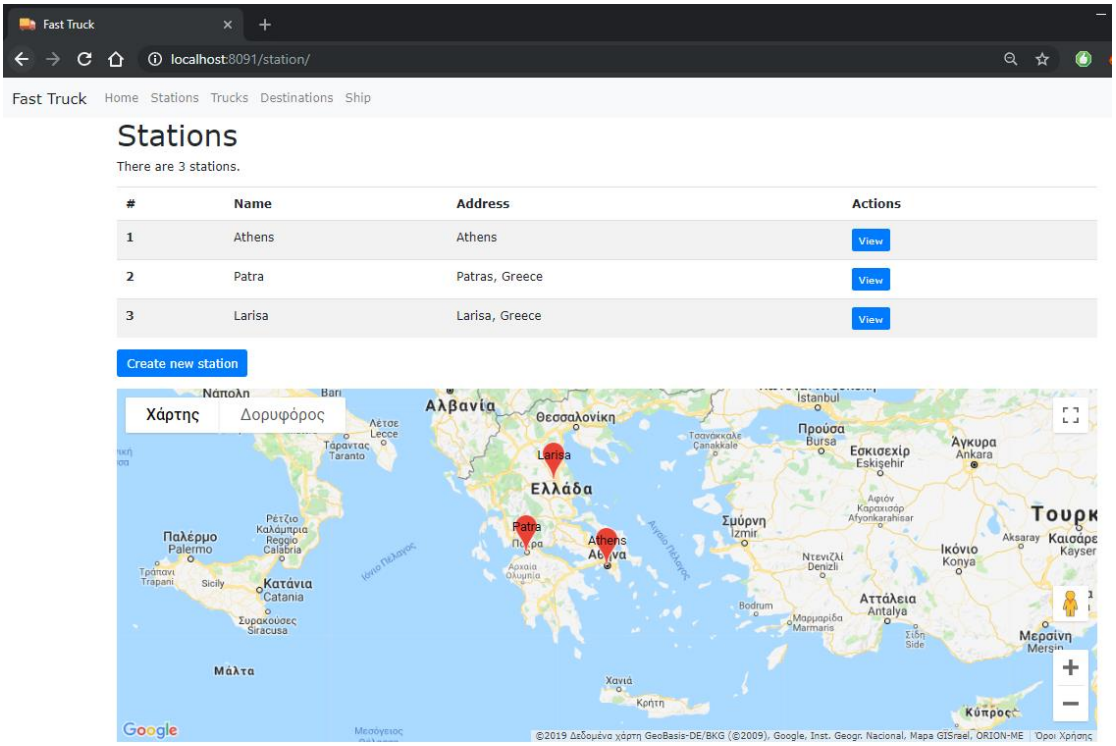
Εικόνα 19: Σελίδα Trucks.

Αν πατήσουμε το κουμπί View σε κάποιο από τα φορτηγά, τότε θα μεταφερθούμε σε νέα σελίδα που μας παρουσιάζει τις πληροφορίες για το συγκεκριμένο φορτηγό, καθώς και τις ενέργειες που μπορούμε να κάνουμε για αυτό. Το περιεχόμενο αυτό παρουσιάζεται στην Εικόνα 20. Οι ενέργειες που μπορούμε να κάνουμε σε ένα φορτηγό είναι να αλλάξουμε την ονομασία του ακολουθώντας το κουμπί Edit ή να το διαγράψουμε εντελώς ακολουθώντας το κουμπί Delete.



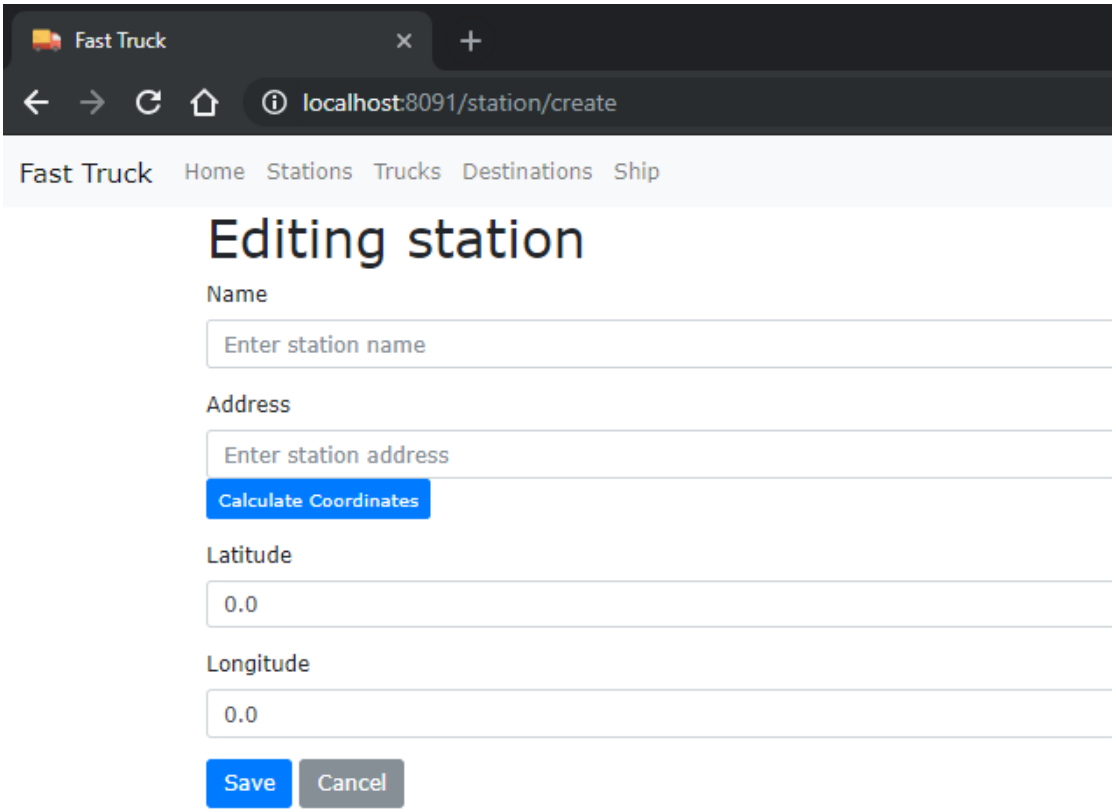
Εικόνα 20: Σελίδα προβολής φορτηγού.

Ακολουθώντας το σύνδεσμο Stations βλέπουμε τις πληροφορίες για τους σταθμούς που υπάρχουν στο σύστημα, όπως φαίνεται στην Εικόνα 21: Σελίδα Stations.Εικόνα 19. Επιπλέον των πληροφοριών στη μορφή πίνακα, όπως προηγουμένως, τοποθετούμε τους υπάρχοντες σταθμούς και σε χάρτη Google Maps για την καλύτερη απεικόνιση στο χρήστη. Στη σελίδα αυτή μπορούμε να πλοηγηθούμε περαιτέρω για να δούμε περισσότερες πληροφορίες για συγκεκριμένο σταθμό, είτε να δημιουργήσουμε νέο σταθμό.



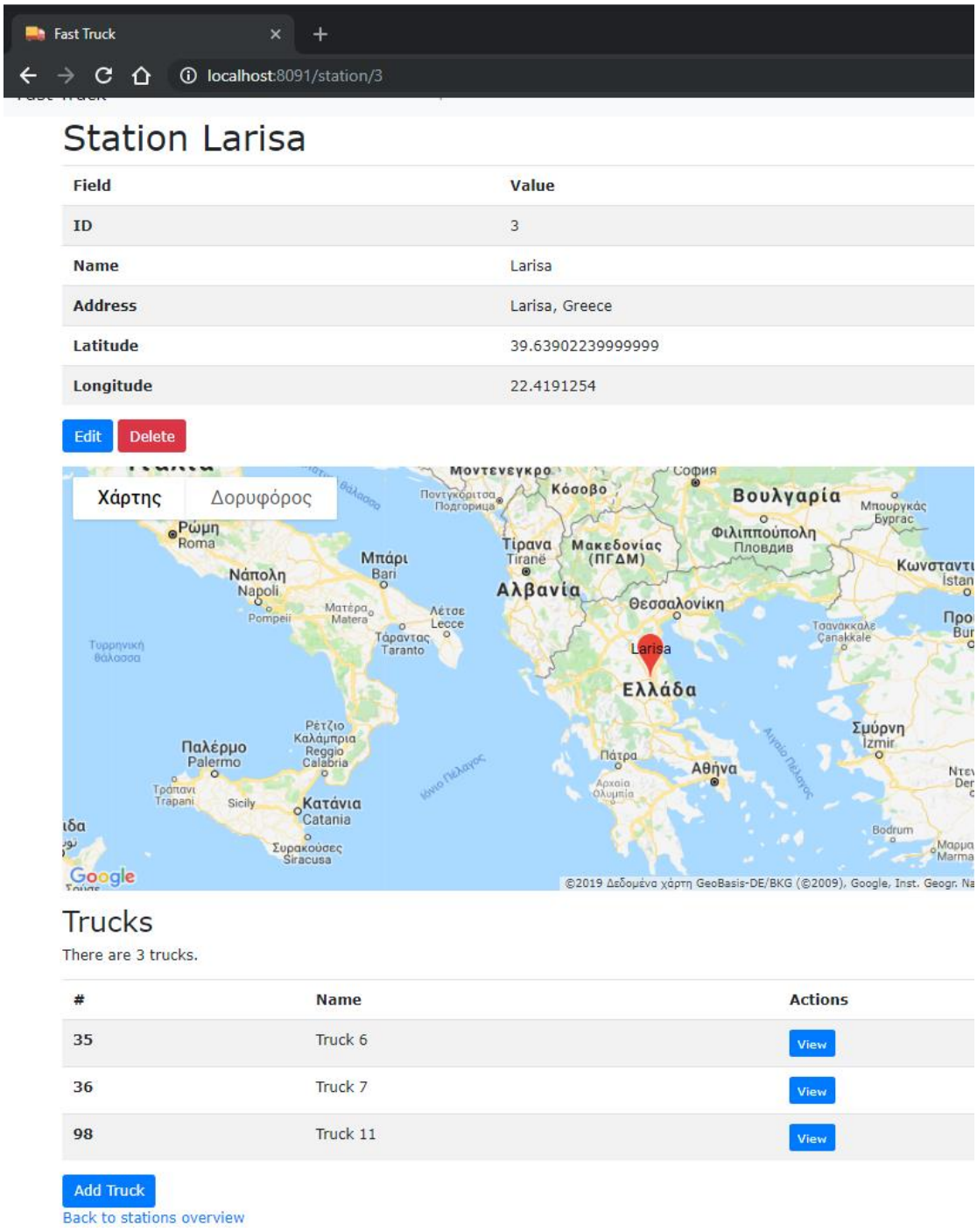
Εικόνα 21: Σελίδα Stations.

Ακολουθώντας το κουμπί «Create new station» μεταφερόμαστε στη σελίδα δημιουργίας νέου σταθμού που φαίνεται στην Εικόνα 22. Η σελίδα είναι η ίδια που χρησιμοποιείται και για την τροποποίηση ενός υπάρχοντος σταθμού. Ο χρήστης πρέπει να εισάγει το όνομα του σταθμού και τη διεύθυνση του. Έπειτα, πατώντας το κουμπί «Calculate Coordinates» υπολογίζονται αυτόματα οι συντεταγμένες της διεύθυνσης μέσω του Google Maps API και εμφανίζονται στα πεδία Latitude και Longitude. Για την καταχώρηση του σταθμού στη βάση δεδομένων, ο χρήστης πρέπει να πατήσει το κουμπί Save.



Εικόνα 22: Δημιουργία νέου σταθμού.

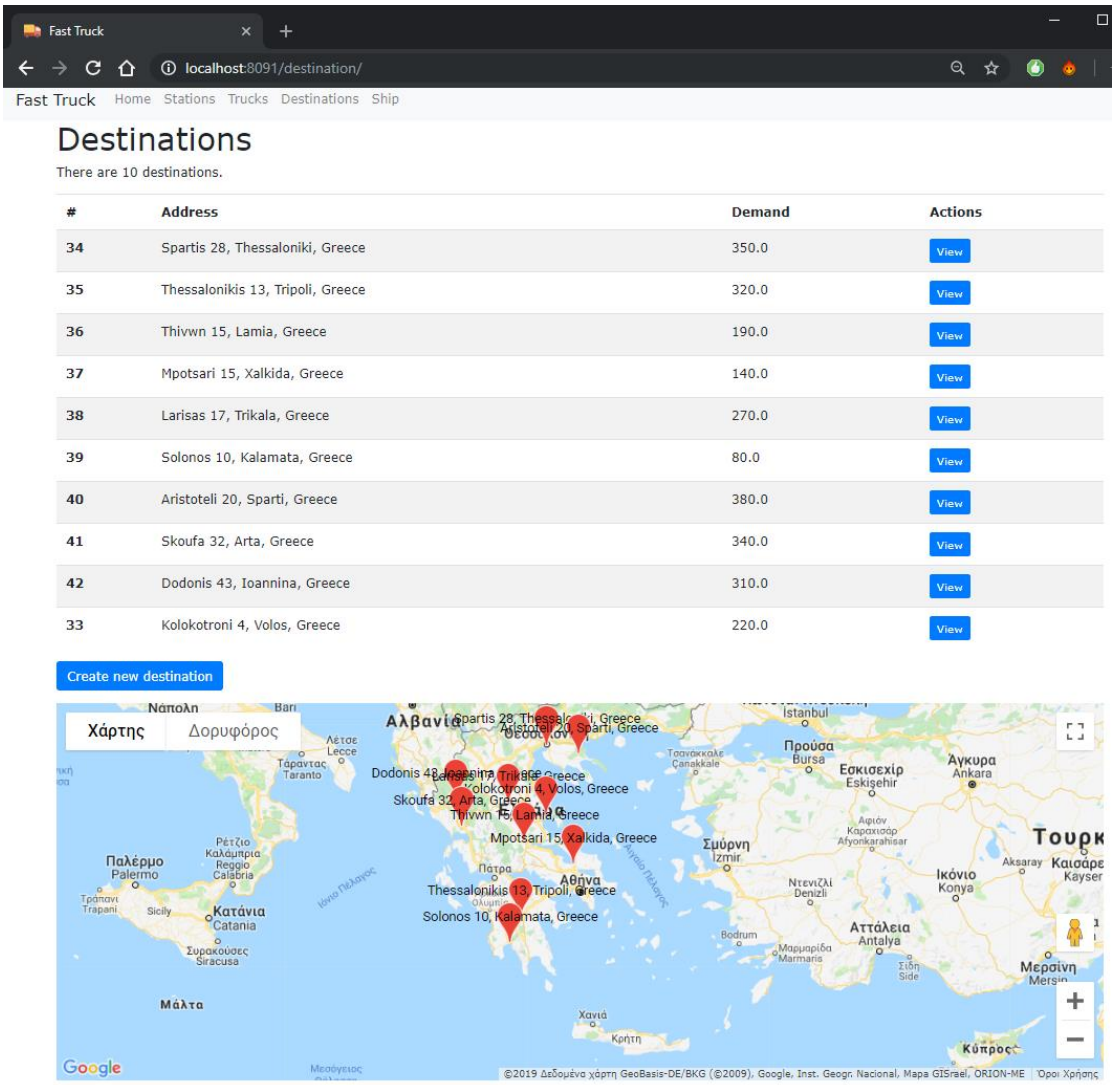
Πατώντας το κουμπί View για συγκεκριμένο σταθμό, μεταφερόμαστε στη σελίδα που παρουσιάζει όλες τις σχετικές πληροφορίες κι ενέργειες για το σταθμό, όπως φαίνεται στην Εικόνα 23. Στη σελίδα αυτή παρουσιάζονται σε μορφή πίνακα οι πληροφορίες που είναι καταχωρημένες στη βάση δεδομένων. Επιπλέον, εμφανίζεται χάρτης που έχει ένα pin στο σημείο του σταθμού. Ο χρήστης μπορεί να τροποποιήσει κάποιο από τα στοιχεία του σταθμού, πατώντας το κουμπί Edit ή να τον διαγράψει από τη βάση δεδομένων πατώντας το κουμπί Delete. Τέλος, εμφανίζονται και πληροφορίες για τα φορτηγά που ανήκουν μόνο στο συγκεκριμένο σταθμό, ενώ ο χρήστης μπορεί να μεταφερθεί στη σελίδα συγκεκριμένου φορτηγού ή να προσθέσει ένα φορτηγό στο συγκεκριμένο σταθμό, πατώντας το κουμπί «Add Truck» και εισάγοντας τα απαραίτητα στοιχεία.



Εικόνα 23: Προβολή σταθμού.

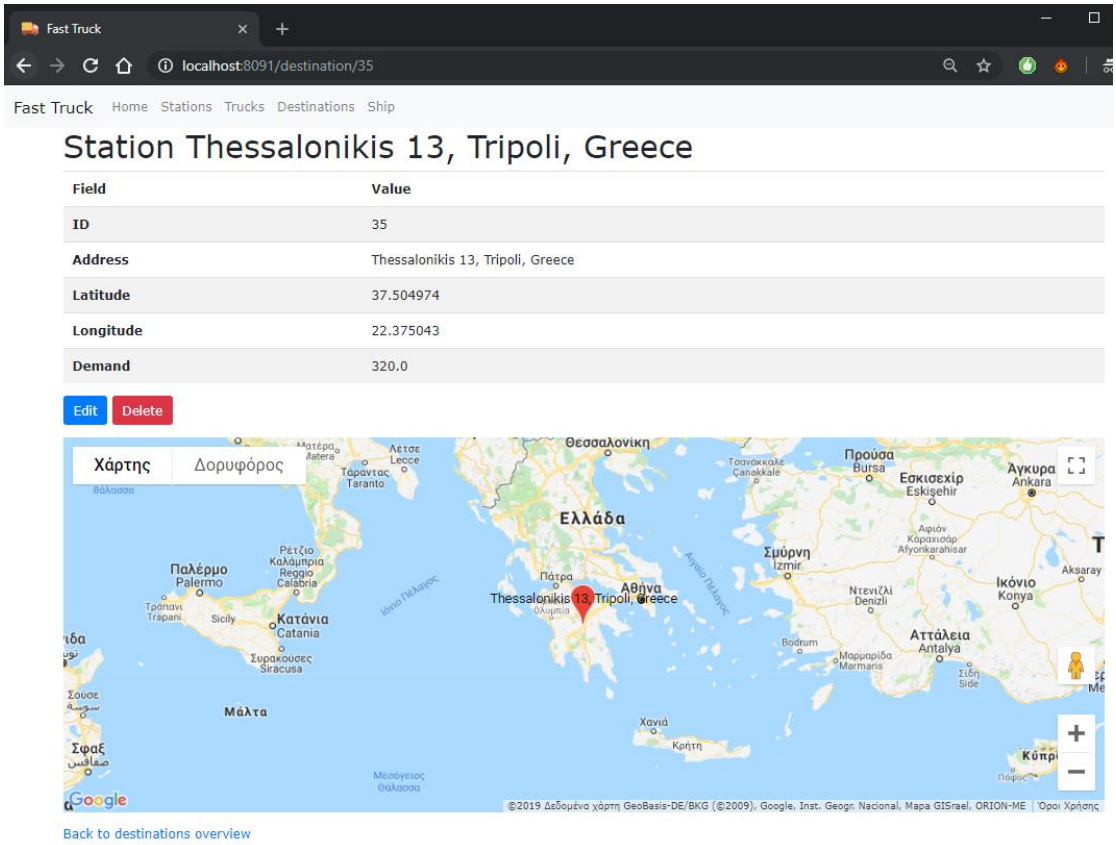
Ακολουθώντας το σύνδεσμο Destinations βλέπουμε τις πληροφορίες για τους προορισμούς που υπάρχουν στο σύστημα, όπως φαίνεται στην Εικόνα 24Εικόνα 19. Επιπλέον των πληροφοριών που υπάρχουν στη βάση δεδομένων στη μορφή πίνακα τοποθετούμε τους υπάρχοντες προορισμούς και σε χάρτη Google Maps για την

καλύτερη απεικόνιση στο χρήστη. Στη σελίδα αυτή μπορούμε να πλοηγηθούμε περαιτέρω για να δούμε περισσότερες πληροφορίες για συγκεκριμένο προορισμό, είτε να προσθέσουμε νέο προορισμό στο πρόβλημα.



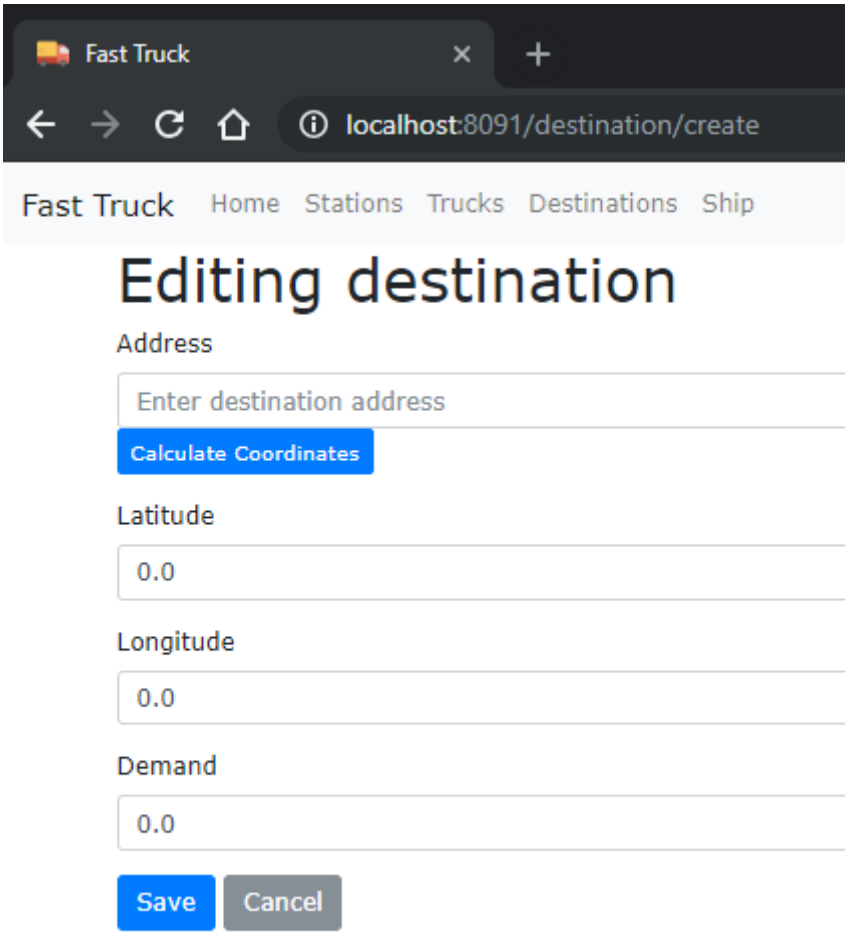
Εικόνα 24: Σελίδα Destinations.

Πατώντας το κουμπί View σε συγκεκριμένο προορισμό βλέπουμε όλες τις σχετικές πληροφορίες που υπάρχουν για αυτόν όπως φαίνεται στην Εικόνα 25. Ο χρήστης μπορεί να τροποποιήσει τα στοιχεία του προορισμού πατώντας το κουμπί Edit ή να τον αφαιρέσει από το πρόβλημα πατώντας το κουμπί Delete.



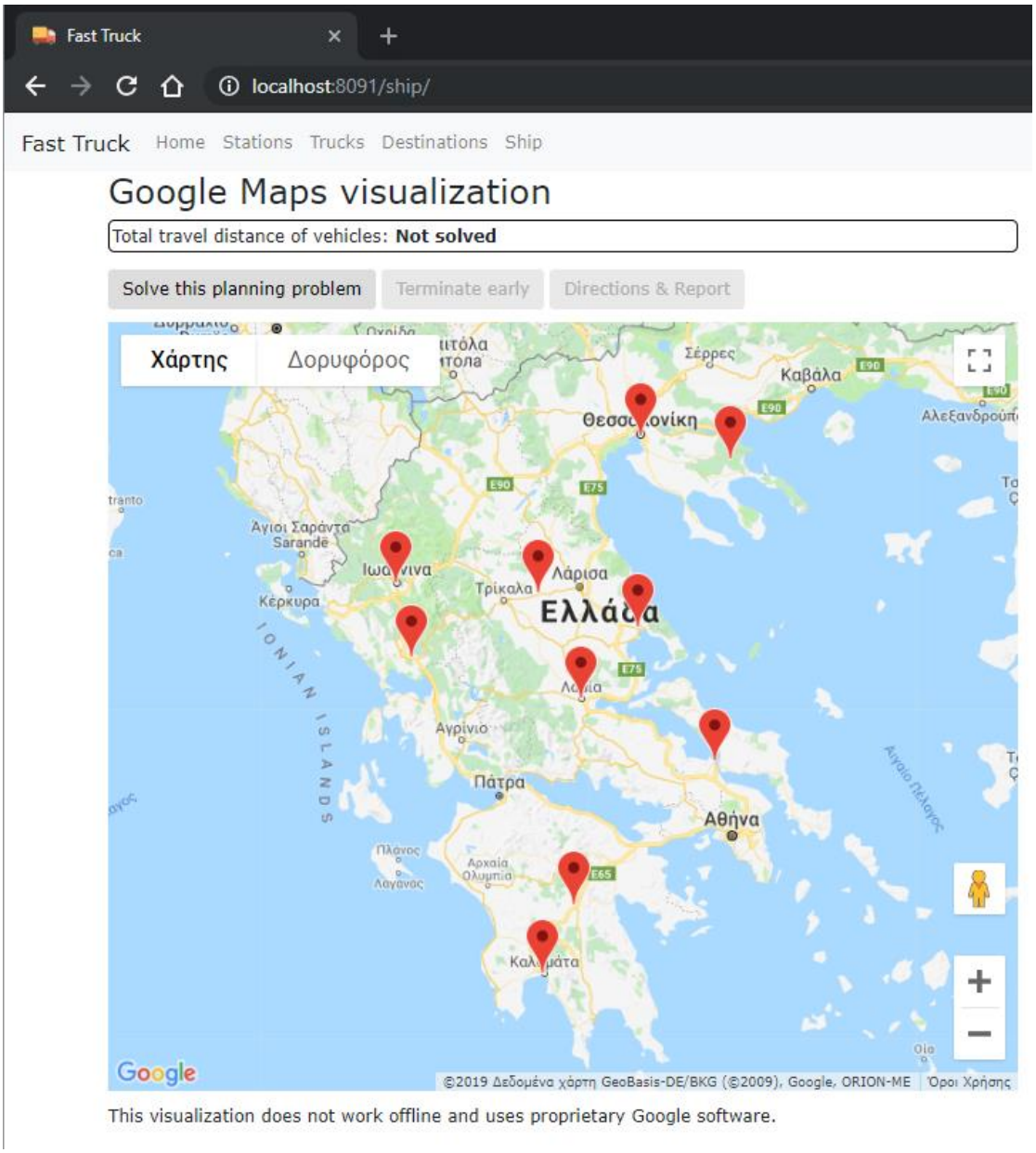
Εικόνα 25: Προβολή προορισμού.

Αν ο χρήστης επιλέξει να δημιουργήσει ένα νέο προορισμό στη σελίδα της Εικόνα 24, τότε θα μεταφερθεί στη σελίδα που παρουσιάζεται στην Εικόνα 26. Ο χρήστης εισάγει και πάλι μόνο τη διεύθυνση και οι γεωγραφικές συντεταγμένες υπολογίζονται αυτόματα. Ο χρήστης πρέπει επιπλέον να προσθέσει τις απαιτήσεις του προορισμού και να πατήσει το κουμπί Save για την αποθήκευση του προορισμού.



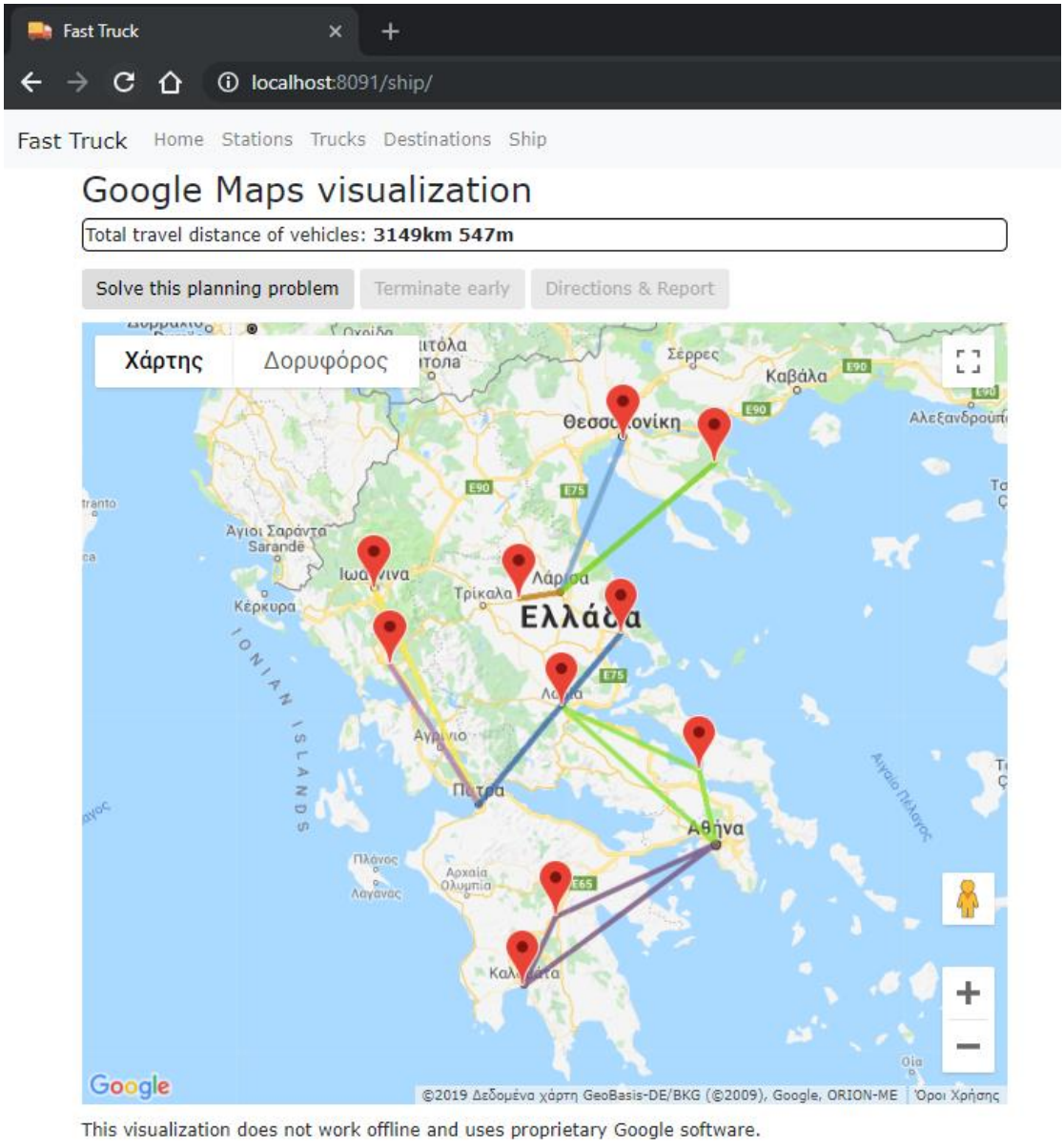
Εικόνα 26: Σελίδα δημιουργίας προορισμού.

Εφόσον λοιπόν ο χρήστης έχει δημιουργήσει τους σταθμούς που θέλει να εξυπηρετήσει για τις ανάγκες του προβλήματος επιλέγει να μεταφερθεί στην σελίδα του ship.



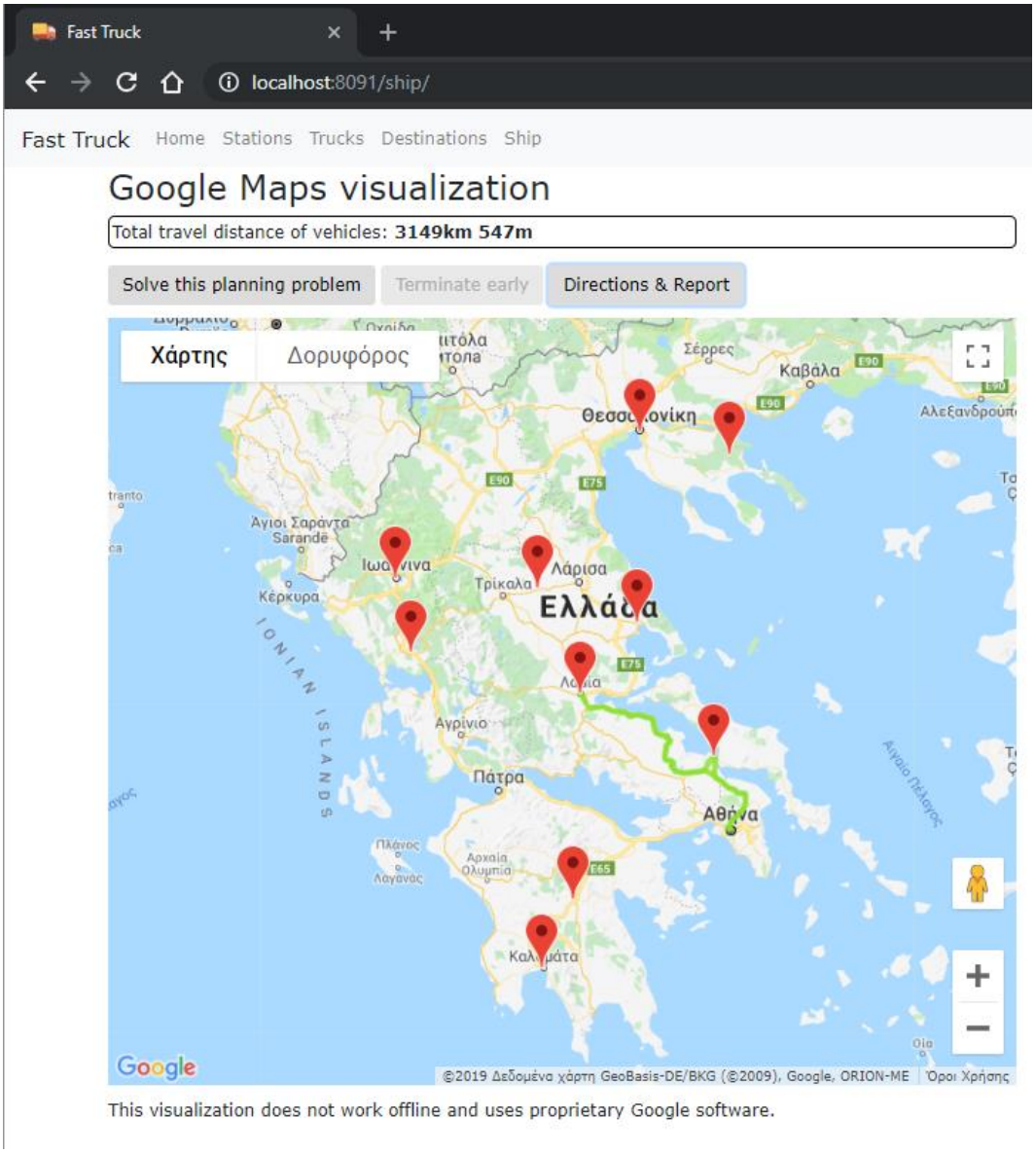
Εικόνα 27: Σελίδα Ship.

Όταν έχει φορτωθεί η σελίδα του Ship, ο χρήστης βλέπει στο χάρτη τους προορισμούς που έχουν οριστεί νωρίτερα και αν πατήσει το κουμπί «Solve this planning problem» θα καλέσει το service της βιβλιοθήκης Optaplanner για την επίλυση του προβλήματος. Όταν ο αλγόριθμος επιλύσει το πρόβλημα, τότε εμφανίζεται η σελίδα που παρουσιάζεται στην Εικόνα 28: Λύση του προβλήματος. και δείχνει τις διαδρομές σε ευθείες γραμμές από κάθε σταθμό προς τους προορισμούς. Ο χρόνος επίλυσης αυξάνεται με το μέγεθος του προβλήματος. Το ίδιο χρώμα γραμμής αντιστοιχεί στο ίδιο φορτηγό. Για παράδειγμα, από την Αθήνα με βάση τις ανάγκες του προβλήματος χρειάστηκαν μόνο δύο φορτηγά για τη μεταφορά των αγαθών. Επίσης, πάνω από το χάρτη εμφανίζεται η συνολική απόσταση που πρέπει να διανύσουν τα φορτηγά.

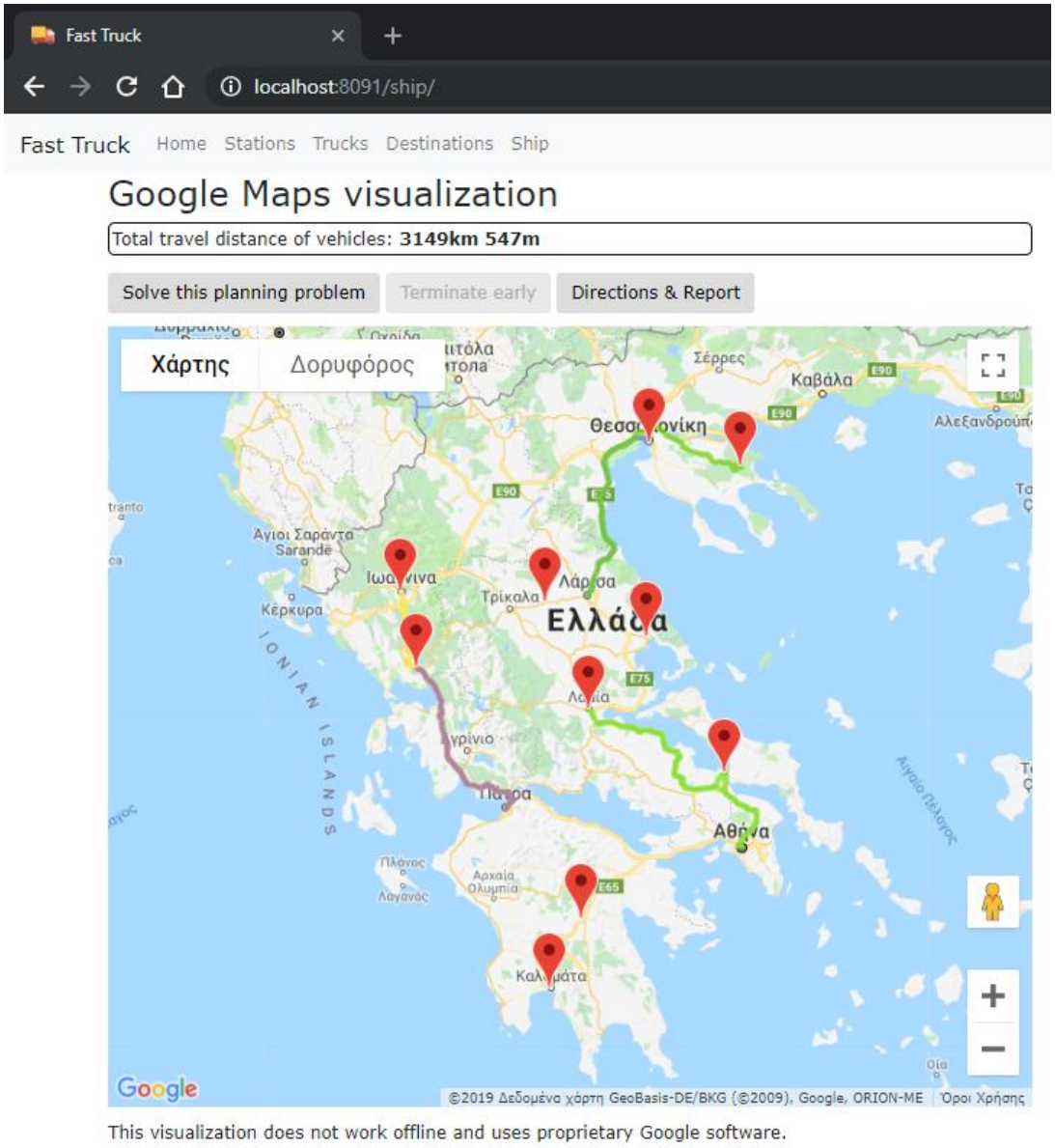


Εικόνα 28: Λύση του προβλήματος.

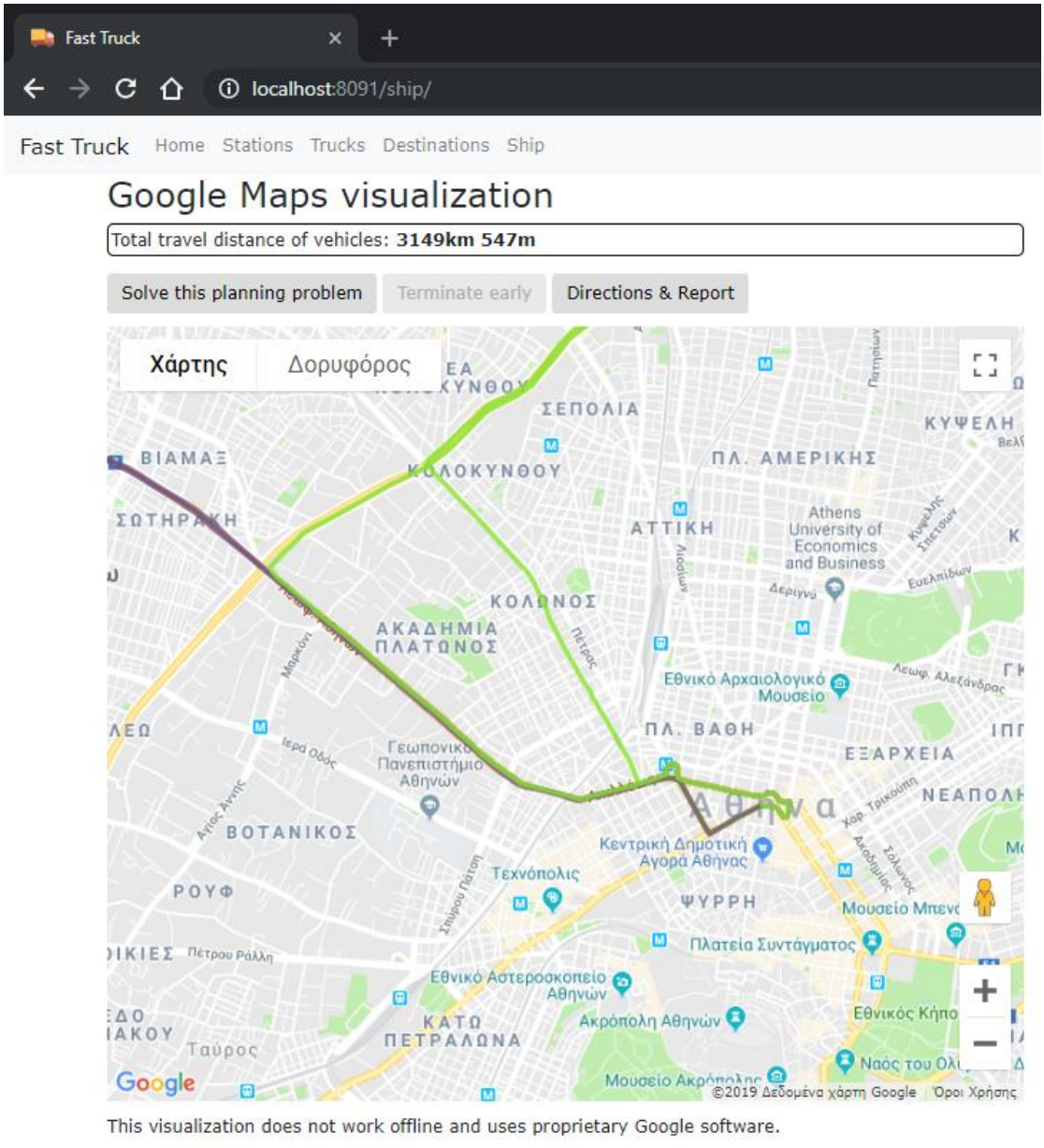
Αν ο χρήστης πατήσει το κουμπί Terminate early ενεργοποιείται και το κουμπί Directions, το οποίο αν το πατήσει ο χρήστης, τότε εμφανίζονται σταδιακά οι πραγματικές διαδρομές που θα εκτελέσουν τα φορτηγά, όπως φαίνεται στις Εικόνα 29, Εικόνα 30, Εικόνα 31



Εικόνα 29: Απεικόνιση αποτελεσμάτων 1.

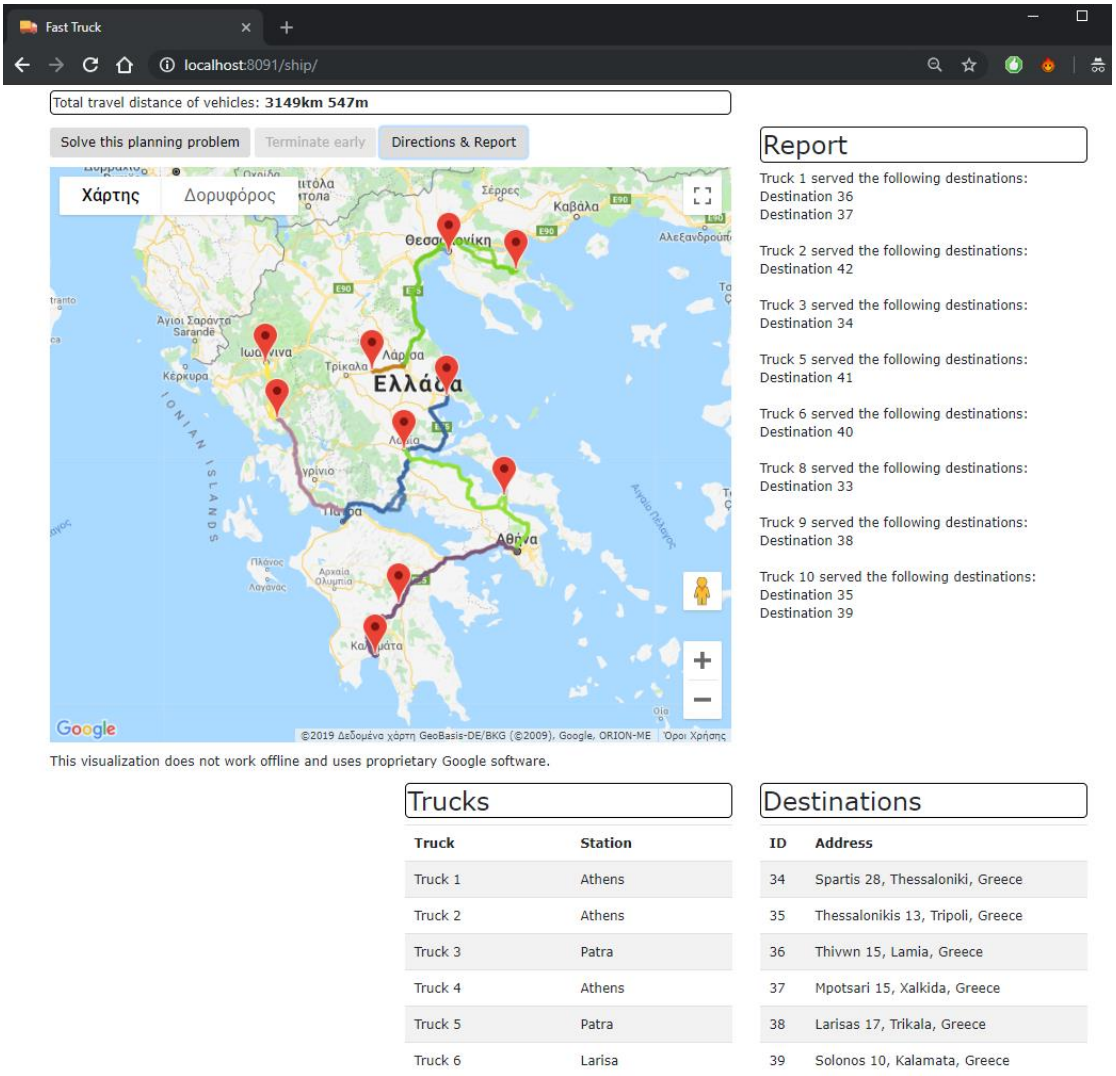


Εικόνα 30: Απεικόνιση αποτελεσμάτων 2.



Εικόνα 31: Απεικόνιση αποτελεσμάτων 3.

Τέλος, ταυτόχρονα με τη διαχάραξη των διαδρομών δίπλα στον χάρτη παρέχεται τρεις πίνακες που παρουσιάζουν ποιους προορισμούς καλύπτει το κάθε φορτηγό όπως φαίνεται και στην Εικόνα 32.



Εικόνα 32: Αναφορά αποτελεσμάτων.

Κεφάλαιο 6

6.1 Συμπεράσματα

Στα πλαίσια της παρούσας εργασίας υλοποιήθηκε μία διαδικτυακή εφαρμογή χρησιμοποιώντας κυρίως τις τεχνολογίες Java, Spring Framework και Optaplanner για τη διαχείριση και την επίλυση του προβλήματος Vehicle Routing Problem with Pickup and Delivery. Οι παράμετροι του προβλήματος μπορούν να μεταβληθούν μέσα από την εφαρμογή, ενώ η απεικόνιση των αποτελεσμάτων γίνεται και χάρτες Google Maps.

Κατά τη διάρκεια της ανάπτυξης της εφαρμογής αποκαλύφθηκε η δυσκολία άμεσης ενσωμάτωσης της βιβλιοθήκης Optaplanner σε ένα project που έχει χτιστεί χρησιμοποιώντας το Spring Framewok. Λόγω της δυσκολίας αυτής, ο συγγραφέας της εργασίας ανέπτυξε περαιτέρω τις ικανότητες του στη διαχείριση προβλημάτων και εύρεση εναλλακτικών λύσεων. Η τελική υλοποίηση της εφαρμογής ήταν επιτυχής με πλήρης λειτουργικότητα. Μία τέτοια εφαρμογή θα μπορούσε να χρησιμοποιηθεί στην πράξη από μικρές και μεγάλες μεταφορικές εταιρίες για τη βελτιστοποίηση των υπηρεσιών τους κι ελαχιστοποίηση του κόστους λειτουργίας τους.

6.2 Μελλοντικές Επεκτάσεις

Λόγω των δυσκολιών που παρουσιάστηκαν κατά την διάρκεια ανάπτυξης της εφαρμογής δεν κατέστη εφικτό να υλοποιηθούν όλες αυτές οι απαιτήσεις που θα έκαναν την συγκεκριμένη εφαρμογή ακόμα πιο εύχρηστη και λειτουργική. Παρακάτω προτείνονται κάποιες βελτιώσεις, επεκτάσεις που θα μπορούσαν να υλοποιηθούν στο μέλλον:

- Προσθήκη και υπολογισμός προβλεπόμενου χρόνου διεκπεραίωσης των δρομολογίων
- Προσαρμογή του προκαθορισμένου ορίου ‘απαιτήσεων’(demand) των προορισμών
- Προσθήκη ξεχωριστών πεδίων pick up, dellivery για κάθε προορισμό(destination)
- Ανάπτυξη κινητής εφαρμογής και δυνατότητα ζωντανής απεικόνισης των δρομολογίων

Βιβλιογραφία

Dhrubojyoti, K. (2008). *Pro Java EE Spring Patterns.*

Eclpse. (n.d.). *Eclipse*. Retrieved 2017, from Eclipse: https://www.eclipse.org

G. B. Dantzig, J. H. (1959). The Truck Dispatching Problem. *Management Science*, 80-91.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 533-549.

Johnson, R. (2004). *The Spring Framework–Reference Documentation.*

Khachaturyan, A. (1979). Statistical-Thermodynamic Approach to Determination of Structure Amplitude Phases. *Sov.Phys. Crystallography*, 519-524.

Lawler. (1985). *The Traveling Salesman Problem.*

M. Ankerst, M. B. (1999). Optics: Ordering Points to Identify the Clustering Structure. *SIGMOD*, (pp. 49-60).

MacQueen, J. (1967). Some Methods for classification and Analysis of Multivariate Observations. *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, (pp. 281-297).

Martin Ester, H.-P. K. (1996). A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, (pp. 226-231). Portland.

Miller, F. P. (2010). Apache.

OpenMP. (n.d.). *OpenMP*. Retrieved 2017, from OpenMP: http://www.openmp.org

OptaPlanner. (2017). *OptaPlanner*. Retrieved from OptaPlanner: https://www.optaplanner.org

Oracle. (2017). *Github*. Retrieved from Oracle: https://github.com/javaee/javaee-spec/blob/master/download/JavaEE8_Platform_Spec_FinalRelease.pdf

Oracle. (2017). *Oracle*. Retrieved from Oracle: http://www.oracle.com/technetwork/java/javaee/overview/index.html

Oracle. (2017). *Oracle*. Retrieved from Oracle: http://www.oracle.com/technetwork/java/intro-141325.html

Ritchie, D. (1993). The Development of the C Language. *ACM SIGPLAN Notices*, 201-208.

Route Optimization API. (n.d.). Retrieved from graphhopper.com: https://graphhopper.com/api/1/docs/route-optimization/

Spring. (2018). *Spring*. Retrieved from http://spring.io/projects/spring-framework

Toth, P. (2001). *The Vehicle Routing Problem.* SIAM.

Vidal, T. (2014). A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, 658–673.

Wei Wang, J. Y. (1997). STING: A statistical information grid approach to spatial data mining. *roceedings of the 23rd International Conference on Very Large Data Bases*, (pp. 186-195).

Λυκοθανάσης. (2017). Εισαγωγή στις Ευρετικές Μεθόδους.

Μπαλαράς, Γ. (2016). Εφαρμογή του αλγορίθμου Tabu Search για την επίλυση του προβλήματος βέλτιστου σχεδιασμού γραμμής προϊόντων.

Evans Ian (2012), Your First Cup: An Introduction to the Java EE Platform

Goncalves Antonio (2013), Beginning Java EE 7, Apress

Σπυρίδων Ξηροκώστας(2011), Εξελικτικός αλγόριθμος για την εκπαίδευση και τη βελτιστοποίηση του μοντέλου των ασαφών γνωστικών απεικονήσεων και των Νευρωνικών Δικτύων

https://docs.oracle.com/javaee/7/JEETT.pdf - Java Platform, Enterprise Edition the Java EE Tutorial, Release 7

http://courses.coreservlets.com/Course-Materials/csajsp2.html - Building Web Apps in Java: Beginning & Intermediate Servlet & JSP Tutorials

https://github.com/kiegroup/optaplanner - Optaplanner, open source framework to optimize the Vehicle Routing Problem

M. David and O. Bräysy, “Active Guided Evolution Strategies for Large-Scale Vehicle Routing Problems with Time Windows,” Computers & Operations Research, Vol. 32, No. 6, 2005

M. Gendreau and J.-Y. Potvin, “Dynamic vehicle routing and dispatching,” in T.G. Crainic and G. Laporte (eds.), Fleet Management and Logistic, pp. 115–226, 1998

Vehicle Routing: Problems, Methods, and Applications, 2nd Edition (MOS-SIAM Series on Optimization)- Edited by Paolo Toth and Daniele Vigo

Δημήτρης Παπαδάκης(1995), Αλγόριθμοι τοπικής αναζήτησης με απαγόρευση κινήσεων και παραλληλοποιηση της αναζήτησης για την επίλυση του προβλήματος ροικής παραγωγής

https://pdfs.semanticscholar.org/fcff/394938381091eccc1c346373a7b5adccc06b.pdf - Solution To Multi-Depot Vehicle Routing Problem Using Genetic Algorithms