ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

CarShare: Αναπτύσσοντας μια πλατφόρμα συνεπιβατισμού με PHP, AngularJS και MongoDB

Θεμιστοκλής Ν. Μπερής

Επιβλέποντες: **Αλέξιος Δελής,** Καθηγητής ΕΚΠΑ
Παναγιώτης Λιάκος, Υποψήφιος Διδάκτωρ

ΑΘΗΝΑ

ΙΟΥΝΙΟΣ 2016

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

CarShare: Αναπτύσσοντας μια πλατφόρμα συνεπιβατισμού με PHP, AngularJS και MongoDB

Θεμιστοκλής Ν. Μπερής
Α.Μ.: 1115201200121

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Αλέξιος Δελής,** Καθηγητής ΕΚΠΑ
Παναγιώτης Λιάκος, Υποψήφιος Διδάκτωρ

ΠΕΡΙΛΗΨΗ

Στην παρούσα πτυχιακή εργασία παρουσιάζουμε την διαδικτυακή πλατφόρμα συνεπιβατισμού **CarShare**. Στόχος της, είναι να μοιράζονται τα έξοδα μιας διαδρομής μέσω του συνεπιβατισμού, τόσο στον χρήστη που παρέχει το αυτοκίνητό του σαν οδηγός, όσο και στον χρήστη που θα χρησιμοποιήσει το αυτοκίνητο σαν επιβάτης.

Αρχικά, παρατίθενται στιγμιότυπα οθόνης (*screenshots*), για να περιγραφεί όλη η λειτουργικότητα του *CarShare*. Σε αυτά περιλαμβάνεται μια αρχική όψη του *CarShare*, η Είσοδος στο σύστημα, η Καταχώρηση Διαδρομής, ένα παράδειγμα αναζήτησης διαδρομής και παρουσίασης των αποτελεσμάτων αναζήτησης καθώς και η Αποσύνδεση.

Στην συνέχεια, παρουσιάζεται ο αλγόριθμος **Longest Common Subsequence (LCSS)**, τον οποίο χρησιμοποιεί το *CarShare* για να υπολογίσει την ομοιότητα μεταξύ δύο διαδρομών και εξηγούμε τους λόγους που τον προτιμήσαμε σε σχέση με άλλους, ήδη υπάρχοντες αλγορίθμους και περιγράφουμε αναλυτικά τον αλγόριθμο.

Έπειτα αναλύουμε τον τρόπο αναπάραστασης των διαδρομών του *CarShare* σε μια *μη σχεσιακή βάση δεδόμενων (NoSQL database)*, την **MongoDB**, και αναφέρονται οι λόγοι για τους οποίους μια τέτοια βάση δεδομένων, απαιτεί ιδανική λύση. Παρουσιάζεται ο αλγόριθμος **RouteBoxer** και ο τρόπος με τον οποίο μπορεί να συνδυαστεί με την *MongoDB*, ώστε να αποκόπτονται διαδρομές που χρησιμοποιούνται σαν είσοδο για εύρεση ομοιότητας στον *LCSS*.

Αναφέρουμε τεχνικά θέματα σχετικά με την υλοποίηση του *CarShare*. Παρουσιάζεται η προσέγγιση της εφαρμογής μας, ώστε να πληρούνται οι απαιτήσεις της αρχιτεκτο-νικής *Representational State Transfer (**REST**)*. Σε αυτά περιλαμβάνεται η χρήση του *Σκελετού Laravel (**Laravel Framework**)* για το *παρασκήνιο (backend)*, η χρήση του *Σκελετού AngularJS (**AngularJS Framework**)* για το *προσκήνιο (frontend)* και ο τρόπος με τον οποίο συνδυάζονται για να αναπτυχθεί το *CarShare* πάνω σε μια αρχιτεκτονική *Μοντέλο-Όψη-Ελεγκτή (**Model-View-Controller (MVC)**)*. Αναφέρουμε τον τρόπο με τον οποίο γίνεται ταυτοποίηση (*authentication*) των χρηστών, ακολουθώντας την *ταυτοποίηση βασισμένη σε λεκτικές μονάδες (**token-based authentication**)*, με την χρήση *διαδικτυακών λεκτικών μονάδων JSON (**JSON web tokens**)*, αποφεύγοντας πλήρως τις *συνεδρίες (**sessions**)*. Επίσης αναφέρεται και το πως συνδυάζεται η χρήση της “εισόδου μέσω Facebook” με όλα τα παραπάνω.

Παρουσιάζουμε *συγκριτικές αξιολογήσεις (**benchmarks**)* για το πώς κλιμακώνεται το *CarShare* με την αύξηση των διαδρομών στην βάση δεδομένων.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Διαδικτυακή Εφαρμογή για Συνεπιβατισμό

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: μετρήσεις ομοιότητας διαδρομών, διαδικτυακή πλατφόρμα, nosql βάσεις, mongodb, διαδικτυακές λεκτικές μονάδες json

ABSTRACT

In this paper, we present the carpooling web application **CarShare**. *CarShare's* goal is to divide the travel expenses among all the occupants of a vehicle (driver and passengers).

First, screenshots are provided, in order to describe the main functionality of *CarShare*. The homepage of the application, the *login* functionality, the entry of a trajectory, an example of the search of a trajectory, as well as, the *logout* functionality, are included in the aforementioned screenshots.

The algorithm of the **Longest Common Subsequence (LCSS)** is presented. *LCSS* algorithm is used, in order to compute the similarity between two trajectories and the reasons why this particular algorithm is used, are explained, compared to previous approaches and *LCSS* is described in detail.

The specific format of the trajectories, in order to be stored to a *non relational (NoSQL) database*, is analyzed, as well as, the reasons why a *non relation database* like *MongoDB* is ideal for *CarShare*. The **RouteBoxer** algorithm is presented, in order to reduce the number of "trajectory candidates" that will be used as LCSS inputs.

Afterwards, technical details about *CarShare's* implementation are demonstrated. *CarShare* follows the *Representational State Transfer (**REST**)* architectural style and for this reason, the **Laravel Framework** and **AngularJS Framework** was used for the backend and frontend of the application, respectively. The combination of the aforementioned *Frameworks* in a specific way, was followed in order to comply with the *Model-View-Controller (**MVC**)* architectural pattern. **JSON web tokens** are introduced, as a way to implement a **token-based authentication** and completely deactivate **PHP sessions** in the application. Moreover, *Login via Facebook* functionality and the way it is combined with the above, are demonstrated.

Finally, benchmarking tests, to examine how scalar our application is, depending on the number of the trajectories stored in the database, are provided.

SUBJECT AREA: Car sharing Web Application

KEYWORDS: trajectory similarity measures, web application, nosql databases, mongodb, json web tokens

Η παρούσα πτυχιακή είναι αφιερωμένη στην οικογένειά μου,
στους γονείς μου Νίκο και Ειρήνη και στην αδερφή μου Δέσποινα.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον κ. Αλέξη Δελή, που μου έδωσε την ευκαιρία να ασχοληθώ με ένα τόσο ενδιαφέρον θέμα, καθώς και για την καθοδήγηση που μου παρείχε για την εκπόνηση της παρούσας πτυχιακής.

Επίσης, θα ήθελα να ευχαριστήσω τον κ. Παναγιώτη Λιάκο, για τις προτάσεις του σε όλα τα στάδια της πτυχιακής μου και για την γενικότερη συνεργασία μας. Η βοήθεια του ήταν καθοριστικής σημασίας για την περαίωση αυτού του έργου.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

1. ΕΙΣΑΓΩΓΗ

Η αύξηση του κόστους μετακίνησης σε συνδυασμό με την κυκλοφοριακή συμφόρηση, έχουν καταστήσει αναγκαία την ύπαρξη μιας εφαρμογής, όπου οι χρήστες θα μπορούν να καταχωρούν διαδρομές ή να επιλέγουν την βέλτιστη για συνεπιβατισμό διαδρομή, μέσα από ένα σύνολο διαθέσιμων διαδρομών. Σκοπός της εφαρμογής είναι να μοιράζονται τα έξοδα της διαδρομής, μεταξύ όλων των εμπλεκόμενων (οδηγός και επιβάτες).

Έτσι, στο πλαίσιο αυτής της πτυχιακής, αναπτύχθηκε η διαδικτυακή εφαρμογή CarShare, όπου χρησιμοποιήθηκαν ποικίλες τεχνολογίες και πρακτικές, με στόχο την αύξηση της αποδοτικότητας της εφαρμογής και την δημιουργία μιας λειτουργικής διεπαφής για τον χρήστη.

Μπορούμε να πούμε ότι η εφαρμογή CarShare, αποτελείται ουσιαστικά από δύο κορμούς. Ο πρώτος κορμός είναι η καταχώρηση της διαδρομής. Αφορά τους χρήστες που διαθέτουν το αυτόκινητό τους και έχουν σκοπό να μοιραστούν με τους συνεπιβάτες τους τα καύσιμα της μετακίνησης. Ο δεύτερος, είναι η αναζήτηση της διαδρομής από τους χρήστες που ψάχνουν μεταφορικό μέσο. Αυτοί αναμένουν να τους εμφανιστούν διαδρομές που θα είναι όσο το δυνατόν περισσότερο όμοιες με την διαδρομή που ψάχνουν.

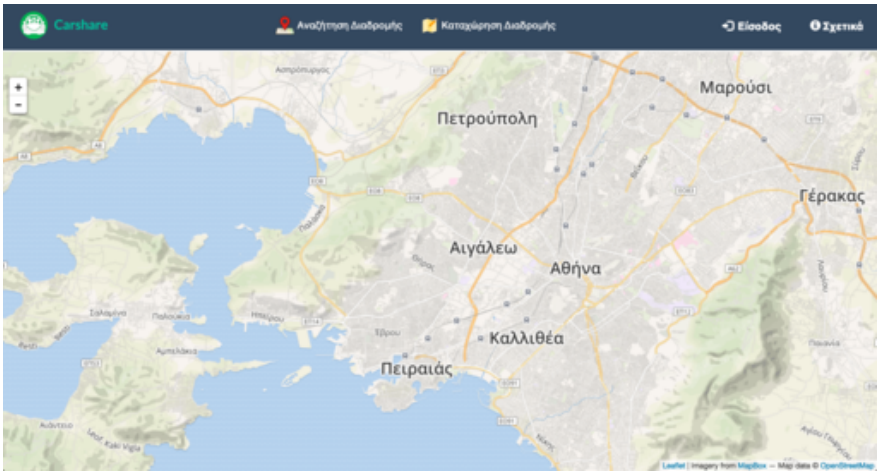
Από τα παραπάνω καταλαβαίνουμε, ότι θα πρέπει να χρησιμοποιηθεί αρχικά ένας αλγόριθμος, όπου δεδομένου δύο διαδρομών, θα προσπαθεί να υπολογίσει την ομοιότητά τους. Στην συνέχεια, είναι προφανές ότι όσο περισσότερες διαδρομές θα καταχωρούνται στην βάση δεδομένων, θα είναι ασύμφορος ο υπολογισμός της ομοιότητας όλων των διαδρομών της βάσης με την υποψήφια προς αναζήτηση διαδρομή. Επομένως θα πρέπει να αναπτυχθεί και ένας επιπλέον αλγόριθμος, όπου θα περιορίζει τις υποψήφιες διαδρομές για τον υπολογισμό της ομοιότητάς τους, ως προς την διαδρομή για αναζήτηση.

2. ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

Στην συνέχεια παρατίθονται στιγμιότυπα της εφαρμογής μας, περιγράφοντας έτσι την βασική της λειτουργικότητα.

2.1 Αρχική Σελίδα

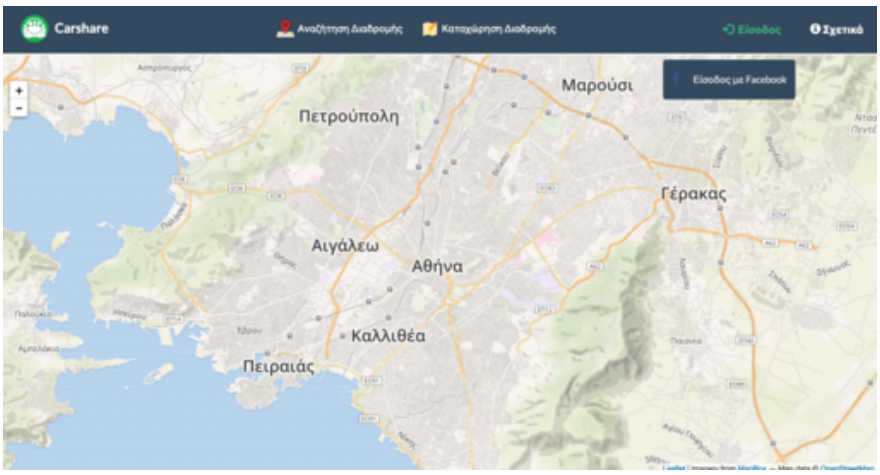
Ο χρήστης στην αρχική σελίδα, βλέπει τις βασικές λειτουργίες που μπορεί να κάνει στο CarShare. Του δίνεται η δυνατότητα να προβεί σε **Αναζήτηση Διαδρομής**, **Καταχώρηση Διαδρομής**, να κάνει **Είσοδο**, η απλά να περιηγηθεί στον χάρτη.



Σχήμα 1: Αρχική Σελίδα

2.2 Είσοδος στο CarShare

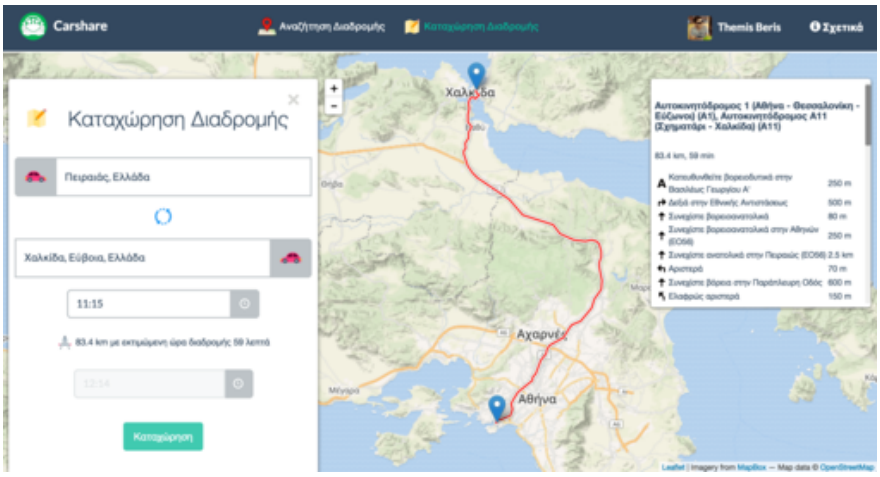
Ο χρήστης προκειμένου να προβεί σε **Καταχώρηση Διαδρομής**, θα πρέπει να έχει κάνει **Είσοδο** στο CarShare. Αυτό γίνεται μέσω του λογιαριασμού του στο Facebook. Το Facebook, είναι το πλέον διαδεδομένο μέσο κοινωνικής δικτύωσης και χρησιμοποιήθηκε σαν τρόπος εισόδου της εφαρμογής μας, ώστε ο χρήστης να μπορεί να σχηματίσει μια εικόνα για τον πιθανό συνεπιβάτη του ή να επικοινωνήσει μαζί του. Έτσι, επιδιώκεται να αναπτυχθεί εμπιστοσύνη μεταξύ των συνεπιβατών πριν αρχίσει το ταξίδι.



Σχήμα 2: Είσοδος μέσω Facebook

2.3 Καταχώρηση Διαδρομής

Κατά την Καταχώρηση Διαδρομής, ο χρήστης αφού συμπληρώσει την **Τοποθεσία Αναχώρησης**, την **Τοποθεσία Άφιξης** και την **Ώρα Αναχώρησης**, βλέπει σε προεπισκόπηση την διαδρομή που θα ακολουθήσει, οδηγίες για την διαδρομή, καθώς και την εκτιμώμενη ώρα άφιξης.



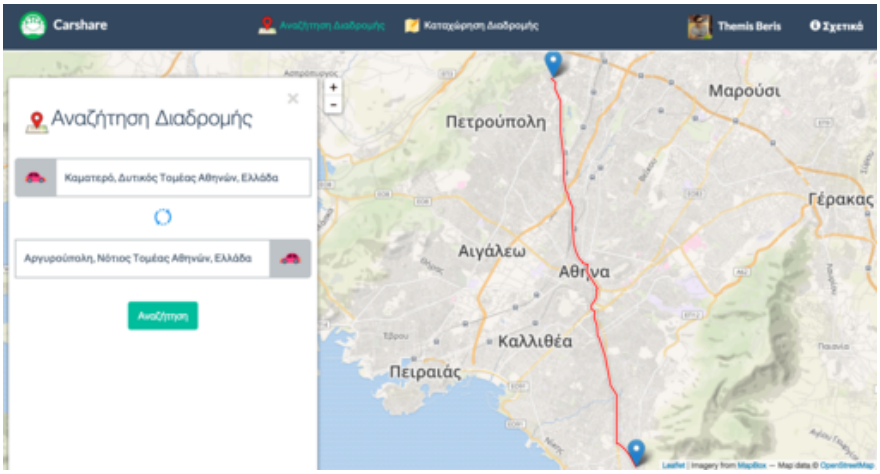
Σχήμα 3: Καταχώρηση Διαδρομής με προεπισκόπηση ταξιδιού

2.4 Αναζήτηση Διαδρομής - Παρουσίαση Αποτελεσμάτων

Στην Αναζήτηση Διαδρομής, ο χρήστης αφού συμπληρώσει την **Τοποθεσία Αναχώρησης** και την **Τοποθεσία Άφιξης**, βλέπει στον χάρτη την διαδρομή που θα ακολουθήσει για να φτάσει στον προορισμό του.

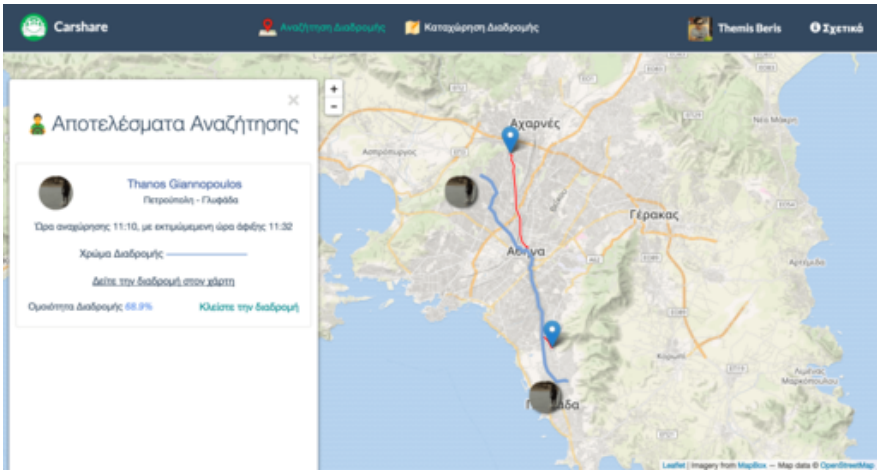
Ας υποθέσουμε λοιπόν, ότι κάποιος χρήστης, έχει κάνει καταχώρηση της διαδρομής *Πετρούπολη - Γλυφάδα* στο CarShare. Ας υποθέσουμε επίσης ότι ενδιαφέρομαστε σαν χρήστες για την διαδρομή *Καματερό - Αργυρούπολη*.

Το πρώτο βήμα θα είναι, όπως αναφέρθηκε, η συμπλήρωση της **Τοποθεσίας Αναχώρησης** και της **Τοποθεσίας Άφιξης**, όπως φαίνεται στην 4.



Σχήμα 4: Αναζήτηση Διαδρομής

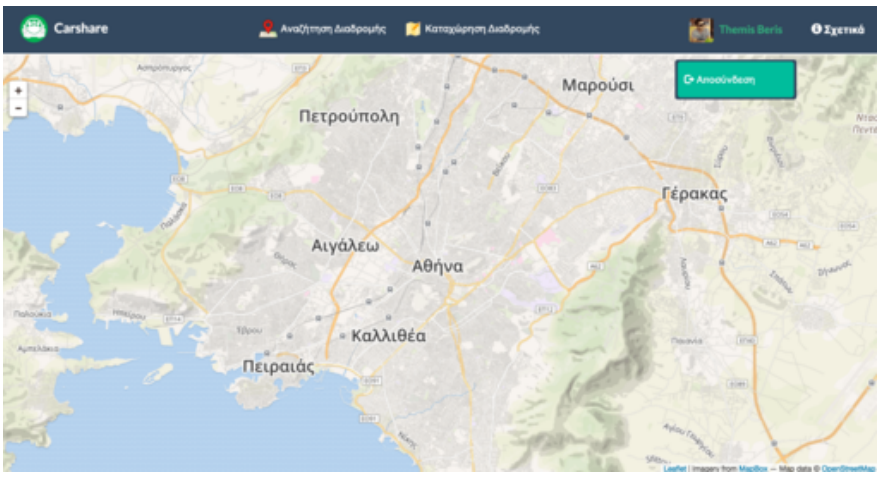
Στην συνέχεια, εμφανίζονται οι διαθέσιμες διαδρομές, όπως φαίνεται στην 5, όπου υπάρχει η δυνατότητα να “κλείσει η διαδρομή” και να έρθουμε σε επικοινωνία με τον χρήστη που καταχώρησε την διαδρομή.



Σχήμα 5: Παρουσίαση Αποτελεσμάτων

2.5 Αποσύνδεση Εφαρμογής

Ο χρήστης που είναι συνδεδεμένος με το CarShare, μπορεί να αποσυνδεθεί από αυτό, όπως φαίνεται στην εικόνα 6 .



Σχήμα 6: Αποσύνδεση από το CarShare

3. ΑΛΓΟΡΙΘΜΙΚΗ ΠΡΟΣΕΓΓΙΣΗ

3.1 Επιλογή του αλγορίθμου LCSS για τον υπολογισμό ομοιότητας διαδρομών

Η επιστήμη της πληροφορικής, έχει ασχοληθεί με την εύρεση ομοιότητας μεταξύ δύο διαδρομών και έχει προτίνει διάφορους αλγορίθμους. Οι πιο γνωστές προσεγγίσεις είναι η χρήση της *Ευκλείδιας Απόστασης* [1] ή της *Δυναμικής Χρονοστρέβλωσης* (*Dynamic Time Warping (DTW)*) [2]. Στο CarShare, υλοποιήθηκε ο αλγόριθμος *LCSS* [3].

Κάθε διαδρομή, μπορεί να αναπαρασταθεί σαν ένα σύνολο από επιμέρους σημεία στον δισδιάστατο χώρο. Το βασικό μειονέκτημα χρήσης της *Ευκλείδιας Απόστασης*, είναι ότι μπορεί να χειριστεί διαδρομές με ίσο πλήθος σημείων. Έτσι, αν θέλαμε να την χρησιμοποιήσουμε, θα έπρεπε να αποφασίσουμε εάν θα αποκόπταμε σημεία από την μεγαλύτερη - σε πλήθος σημείων - διαδρομή ή αν θα γεμίζαμε με σημεία, με μηδενικές συντεταγμένες την μικρότερη διαδρομή. Κάτι τέτοιο, δεν θα λειτουργούσε σωστά, καθώς στην συντριπτική πλειοψηφία οι διαδρομές διαφέρουν ως προς το πλήθος των σημείων, ενώ παράλληλα ο ορισμός της απόστασης γίνεται ασαφής.

Η *DTW*, αποτελεί καλύτερη επιλογή συγκριτικά με την *Ευκλείδια Απόσταση*, εφόσον επιτρέπει την παραμόρφωση της διαδρομής στο πεδίο του χρόνου. Το βασικό μειονέκτημα της *DTW*, είναι ότι πρέπει να ταιριάξει όλα τα σημεία μεταξύ των δύο διαδρομών και επιπλέον δεν μπορεί να υπολογίσει με κατάλληλο τρόπο την ομοιότητα μεταξύ διαδρομών διαφορετικού μήκους [4]. Ειδικά σε περιβάλλον με θόρυβο, παρόλο που κάτι τέτοιο δεν συμβαίνει στην εφαρμογή μας, η *DTW* είναι πιθανόν να κάνει λανθασμένη κατηγοριοποίηση των διαδρομών. Τέλος, στην *DTW*, πρέπει να υπολογίσουμε μια χρονοβόρα L_p *νόρμα*.

Ο *LCSS* είναι ένας αλγόριθμος που μπορεί να υπολογίσει την ομοιότητα διαδρομών διαφορετικού μήκους. Βασικό του πλεονέκτημα αποτελεί το γεγονός ότι δεν είναι απαιτείται ταίριασμα όλων των σημείων των διαδρομών, κάτι που τον καθιστά εύρωστο σε περίπτωση που διαχειριζόμαστε δεδομένα με θόρυβο. Τέλος, με την χρήση του *LCSS*, αποφεύγεται ο υπολογισμός της L_p *νόρμας*, κερδίζοντας έτσι σε απόδοση σε σχέση με τις προηγούμενες προσεγγίσεις που αναφέρθηκαν. Για αυτούς τους λόγους, επιλέχθηκε σαν αλγόριθμος υπολογισμού ομοιότητας μεταξύ διαδρομών για το *CarShare*.

3.2 Ο αλγόριθμος LCSS

Κάθε διαδρομή στο CarShare, μπορεί να αναπαρασταθεί σαν ένα σύνολο από επιμέρους σημεία στον δισδιάστατο χώρο. Δεδομένου δύο διαδρομών A και B, με μέγεθος n και m αντίστοιχα, ορίζεται ένας ακέραιος δ και ένας πραγματικός αριθμός ε, με $0 < \varepsilon < 1$.

Η σταθερά δ, καθορίζει πόσο μακρυά μπορούμε να πάμε στον χρόνο, προκειμένου να ταιριάξουμε ένα σημείο μιας διαδρομής, με ένα σημείο μιας άλλης διαδρομής. Η σταθερά ε, καθορίζει ένα κατώφλι-απόκλιση προκειμένου να ταιριάξουν τα σημεία των διαδρομών.

Ορίζεται η τιμή $LCSS_{\delta,\varepsilon}(A,B)$, ως εξής :

$$LCSS_{\delta,\varepsilon}(A,B) = \begin{cases} 0, & \text{Αν A ή B κενό} \\ 1 + LCSS_{\delta,\varepsilon}(Head(A), Head(B)), & \\ \quad \text{Αν } |a_{x,n} - b_{x,m}| < \varepsilon \text{ και } |a_{y,n} - b_{y,m}| < \varepsilon \text{ και } |n-m| \leq \delta & \\ max(LCSS_{\delta,\varepsilon}(Head(A), B), LCSS_{\delta,\varepsilon}(A, Head(B))), & \text{αλλιώς} \end{cases}$$

Έτσι, υπολογίζοντας πρώτα την $LCSS_{\delta,\varepsilon}(A,B)$, μπορεί να οριστεί η ακόλουθη συνάρτηση ομοιότητας:

$$S(\delta,\varepsilon,A,B) = \frac{LCSS_{\delta,\varepsilon}(A,B)}{min(n,m)}$$

Είναι φανερό ότι ο LCSS, έχει πολυπλοκότητα $O(\delta(n+m))$, πράγμα που σημαίνει ότι ο καθορισμός της σταθεράς δ, παίζει σημαντικό ρόλο στην αποδοτικότητα της εφαρμογής. Στην πράξη, το δ συνίσταται να κυμαίνεται στο $20-30\%$ του μήκους της διαδρομής αναζήτησης [3]. Στο CarShare, ο προσδιορισμός του ε εξαρτάται από τα χιλιόμετρα της διαδρομής προς αναζήτηση. Αν ένας χρήστης ενδιαφέρεται να πραγματοποιήσει μια χιλιομετρικά μεγάλη διαδρομή, είναι πιθανό να συμβιβαστεί με πιο μακρινά σημεία, προκειμένου να πάρει κάποιον συνεπιβάτη, ενώ αν ενδιαφέρεται για μικρές, μέσα στην πόλη αποστάσεις, ο συμβιβασμός αυτός δεν είναι εύκολα εφικτός. Έτσι, το ε είναι ανάλογο του μήκους της διαδρομής.

3.3 Αναπαράσταση Διαδρομών στην Βάση Δεδομένων

Στην εφαρμογή, έχουμε χρησιμοποιήσει μια *NoSQL* βάση δεδομένων, την *MongoDB* [5]. Η επιλογή αυτή έγινε για αρκετούς λόγους. Αρχικά, παρέχει πολύ καλή υποστήριξη για διαχείρηση *γεωγραφικών δεδομένων* (*Geospatial Information*) και σίγουρα είναι πιο ευέλικτη από μια σχεσιακή βάση δεδομένων.

Όντας μια μη-σχεσιακή βάση δεδομένων, δεν κρίνεται αναγκαίο να οριστεί αυστηρά η δομή της βάσης (**schemaless**). Επίσης, η κλιμάκωση της εφαρμογής είναι πολύ εύκολη, καθώς η *MongoDB* μπορεί να κατανεμηθεί σε πολλαπλά κατανεμημένα κέντρα δεδομένων (*data centers*), χωρίς ιδιαίτερη προσπάθεια από τον προγραμματιστή (**sharding**). Εκτός αυτού, προσφερεί την δυνατότητα του **replication**, δηλαδή την δημιουργία πολλαπλών στιγμιοτύπων της βάσης, παρέχοντας έτσι υψηλή διαθεσιμότητα των δεδομένων. Τέλος, σε πολλές περιπτώσεις, αποδεικνύεται ταχύτερη από μια σχεσιακή βάση δεδομένων, τόσο στην εισαγωγή μεγάλων δεδομένων, όσο και στην αναζήτηση [6].

Συγκεκριμένα, οι διαδρομές αποθηκεύονται σε μια *συλλογή* (*collection*), όπου υπάρχει ευρετηρίο *2dsphere* [7]. Ένα τέτοιο ευρετήριο, υποστηρίζει ερωτήματα (*queries*), τα οποία κάνουν γεωμετρικούς υπολογισμούς πάνω σε μια σφαίρα ανάλογης της γης. Για να τεθούν

αυτά τα ερώτηματα στην *MongoDB*, υπαρχούν *τελεστές* (*operators*) που ενσωματώνουν τους γεωμετρικούς υπολογισμούς της *τομής* (*intersection*), *συμπερίληψης* (*inclusion*) και *εγγύτητας* (*proximity*) [9].

Προκειμένου να χρησιμοποιηθεί το ευρετήριο *2dsphere*, πρέπει οι διαδρομές να αποθηκεύονται στην *MongoDB* σαν *GeoJSON* αντικείμενα [8]. Τα *GeoJSON* αντικείμενα, είναι απλά πρότυπα (*formats*), που ακολουθούνται για την αναπαράσταση γεωγραφικών δεδομένων. Τα πιο βασικά από αυτά είναι το *Σημείο* (*Point*), το *LineString*, το Πολύγωνο (*Polygon*) και το *MultiLineString*. Για την αναπαράσταση μιας διαδρομής μπορεί να χρησιμοποιηθεί το *LineString* [10].

Οι *τελεστές* της *MongoDB*, που χρησιμοποιούνται για τα παραπάνω είναι:

1. **$geoWithin**: Επιλέγει αντικείμενα που βρίσκονται γεωμετρικά μέσα στο επερωτούμενο GeoJSON αντικείμενο. Το επερωτούμενο GeoJSON αντικείμενο μπορεί να είναι *Polygon* ή *MultiPolygon*. Υποστηρίζεται από ευρετήρια *2dsphere* ή *2d*. Η χρήση του τελεστή δεν απαιτεί την παρουσία ευρετηρίου. Παρόλα αυτά το ευρετήριο, αυξάνει σημαντικά την απόδοση των ερωτημάτων **(inclusion query)**.
2. **$geoIntersects**: Επιλέγει αντικείμενα που τέμνονται γεωμετρικά με το επερωτούμενο GeoJSON αντικείμενο με την χρήση σφαιρικής γεωμετρίας. Το επερωτούμενο GeoJSON αντικείμενο μπορεί να είναι *Polygon* ή *MultiPolygon*. Υποστηρίζεται μόνο από το ευρετήριο *2dsphere*. Η χρήση του τελεστή δεν απαιτεί την παρουσία ευρετηρίου. Παρόλα αυτά το ευρετήριο, αυξάνει σημαντικά την απόδοση των ερωτημάτων. **(intersection query)**.
3. **$near**: Επιστρέφει αντικείμενα που βρίσκονται γεωμετρικά κοντά σε σχέση με το επερωτούμενο GeoJSON *Σημείο*. Η μέγιστη και η ελάχιστη απόσταση καθορίζεται από τον προγραμματιστή. Τα επιστραμμένα αντικείμενα, είναι ταξινομημένα από το κοντινότερο στο πιο απομακρυσμένο. Υποστηρίζεται από ευρετήρια *2dsphere* ή *2d*. Η χρήση του τελεστή απαιτεί την χρήση ευρετηρίου **(proximity query)**.
4. **$nearSphere**: Κάνει ακριβώς ότι και ο τελεστής *$near*, με την μόνη διαφορά ότι χρησιμοποιείται σφαιρική γεωμετρία **(proximity query)**.

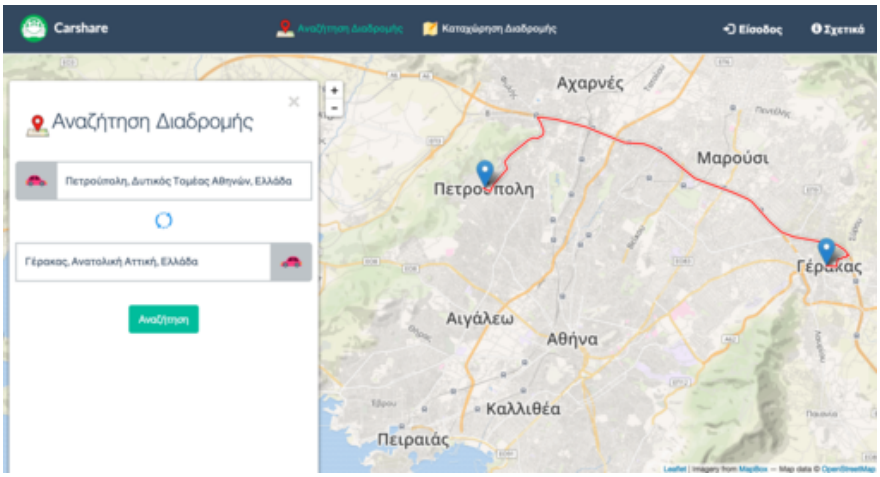
Στο CarShare, μια τυπική διαδρομή που απαρτίζεται από *n* σημεία, αποθηκεύεται σαν το ακόλουθο *LineString* στην βάση δεδομένων:

{
“**type**”: “LineString”,
“**coordinates**”: [[x_1, y_1], [x_2, y_2], ... , [x_n, y_n]]
}

3.4 Μείωση πλήθους διαδρομών για τον LCSS

Στο CarShare, θα ήταν ασύμφορο να υπολογίζεται η ομοιότητα όλων των διαδρομών που υπάρχουν στην βάση, ως προς την διαδρομή που γίνεται αναζήτηση. Μια ιδέα είναι να παραλείπονται οι διαδρομές από τον LCSS, που ούτως ή άλλως θα είχαν μικρή ομοιότητα.

Όπως αναφέρθηκε, ο τελεστής *$geoIntersects*, δέχεται σαν όρισμα ένα GeoJSON πολύγωνο και επιστρέφει όλες τις διαδρομές που βρίσκονται εντός του πολυγώνου. Η διαδρομή που κάνει αναζήτηση ο χρήστης όμως, είναι πρακτικά μια γραμμή (LineString). Για αυτόν τον λόγο, χρησιμοποιείται ο αλγόριθμος *RouteBoxer*, για την μετρατροπή μιας διαδρομής από *LineString* σε *Polygon*.



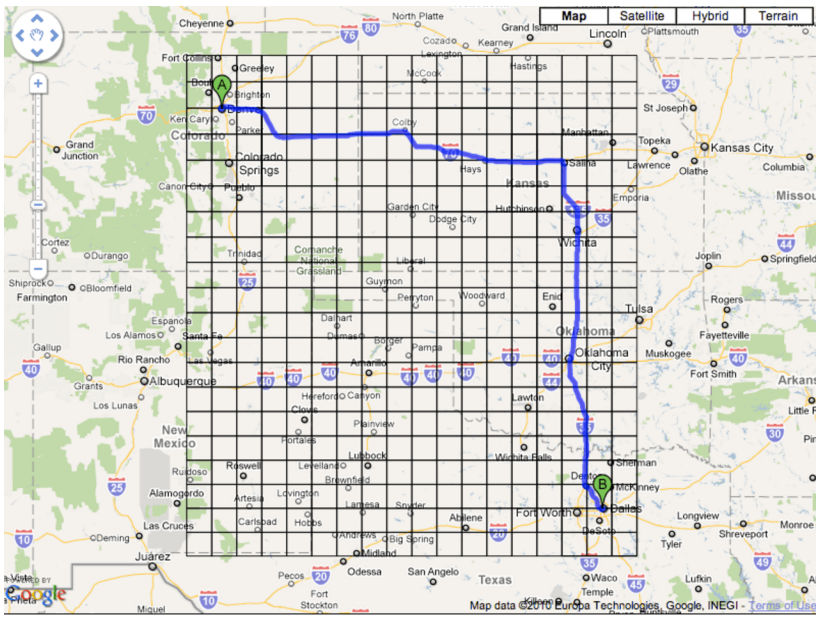
Σχήμα 7: Αναπαράσταση της διαδρομής Πετρούπολη - Γέρακας σαν LineString

3.4.1 Η κλάση RouteBoxer

Το CarShare, επαυξάνει τη διαδρομή-*LineString* χρησιμοποιώντας την **κλάση Routeboxer** (**Routeboxer class**) - αλγόριθμος που αναπτύχθηκε από την Google [11] και χρησιμοποιείται στο *Google Maps API v3*- με την διαφορά ότι τρέχει στο *backend* της εφαρμογής.

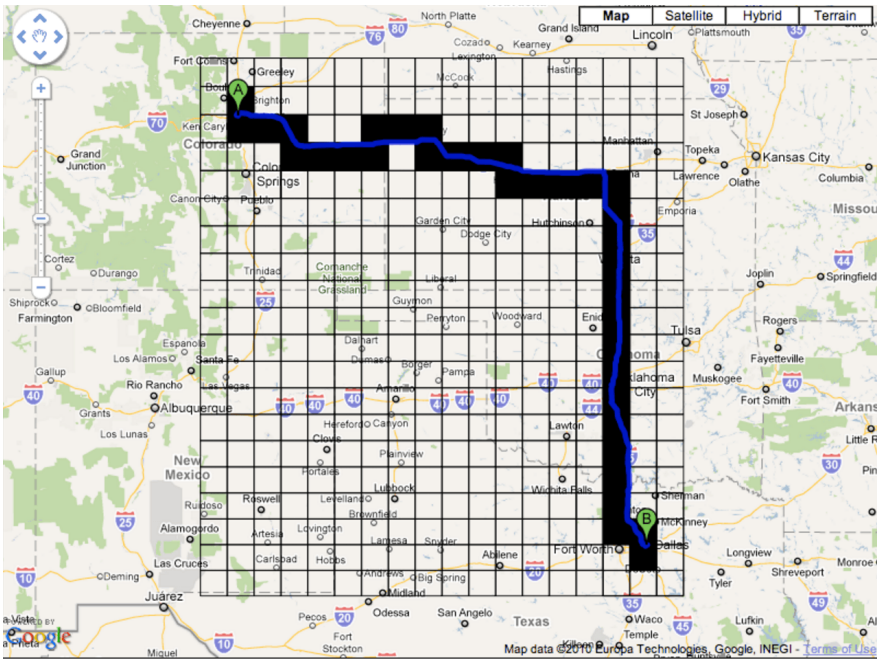
Για την κλάση Routeboxer, ορίζεται ένας αριθμός *d* που αναπαριστά απόσταση σε χιλιόμετρα. Η κλάση μας επιστρέφει ένα σύνολο από παραλληλόγραμμα, τα οποία έχουν την ιδιότητα να καλύπτουν όλα τα επιμέρους σημεία της διαδρομής, μέσα στην ορισμένη απόσταση d. Συνοπτικά, τα βήματα του αλγορίθμου είναι τα ακόλουθα:

1. Τοποθετείται ένα πλέγμα, αποτελούμενο από τετράγωνα κελιά μήκους *d* κατά μήκος της διαδρομής. Το πλέγμα επεκτείνεται περιμετρικά με ένα επιπλέον κελί προς κάθε κατεύθυνση.



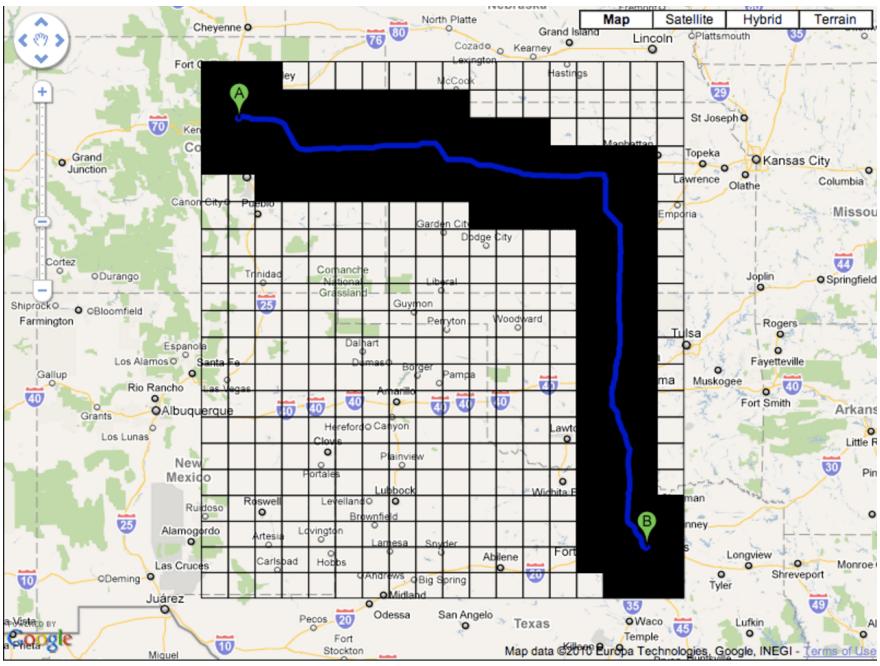
Σχήμα 8: RouteBoxer - Δημιουργία πλέγματος [12]

2. Στην συνέχεια, επισημαίνονται τα κελιά του πλέγματος που τέμνονται με την διαδρομή.



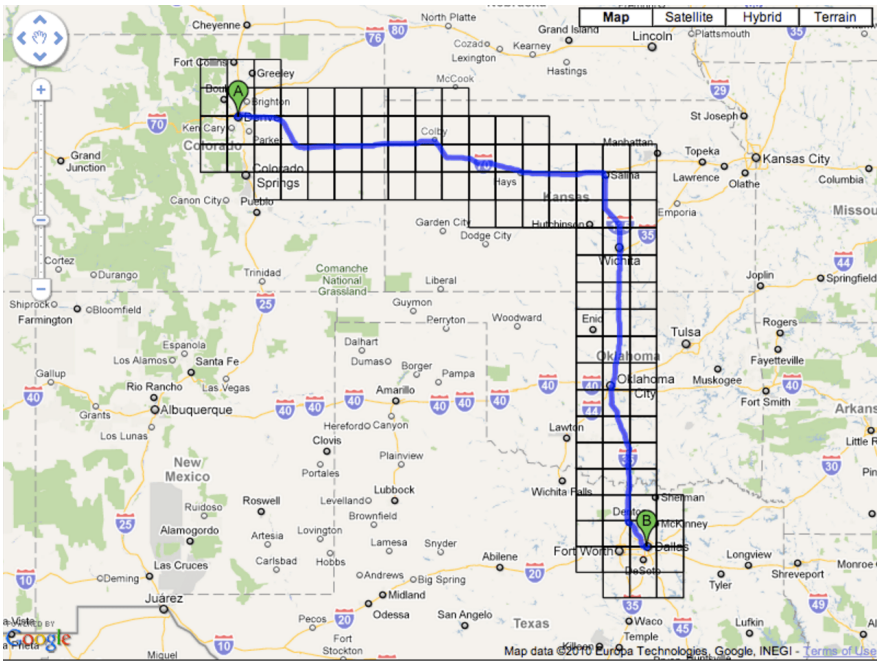
Σχήμα 9: RouteBoxer - Επισημασμένα Κελιά [12]

3. Για κάθε κελί που επισημαίνεται, επισημαίνονται και τα κατευθείαν γειτονικά του κελιά.



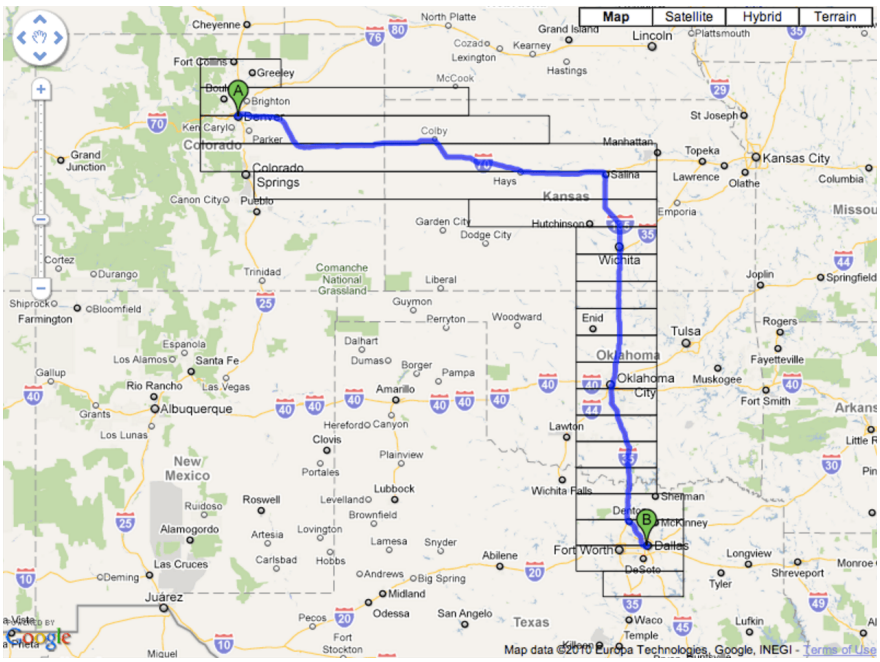
Σχήμα 10: RouteBoxer - Επέκταση Επισημασμένων Κελιών [12]

4. Στην τρέχουσα κατάσταση, κάθε σημείο της διαδρομής, καλύπτεται περιμετρικά με κελιά συνολικής απόστασης *d*.



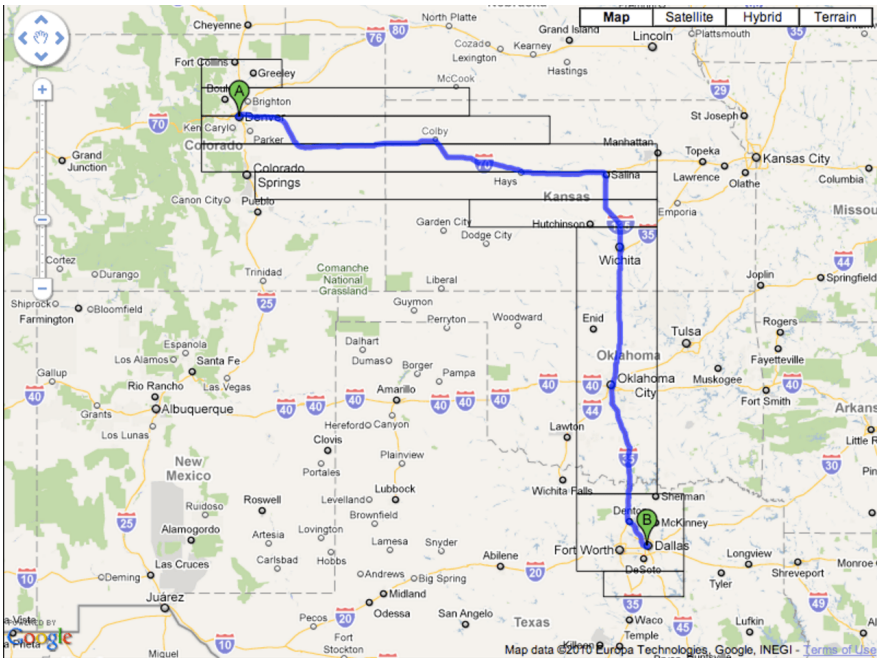
Σχήμα 11: RouteBoxer - Πλέγμα μετά από Αποκοπή Κελιών [12]

5. Έπειτα, μετατρέπονται τα επισημασμένα τετράγωνα σε ένα σύνολο από παραλληλόγραμμα που δεν επικαλύπτονται μεταξύ τους. Σε μια πρώτη προσέγγιση, συγχωνεύονται τα κελιά που είναι **οριζόντια γειτονικά** με άλλα κελιά. Έτσι, δημιουργείται ένα σύνολο r_1 από πλατιά παραλληλόγραμμα, κάθε ένα ύψους *d*.



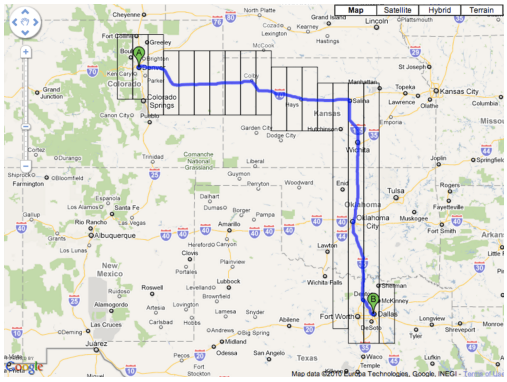
Σχήμα 12: RouteBoxer - Δημιουργία πλατιών παραλληλογράμμων [12]

6. Μετά, κάθε παραλληλόγραμμο, συγκρίνεται με το ακριβώς από κάτω του και αν έχουν ίδιο πλάτος και ίδια οριζόντια θέση, συγχωνεύονται.

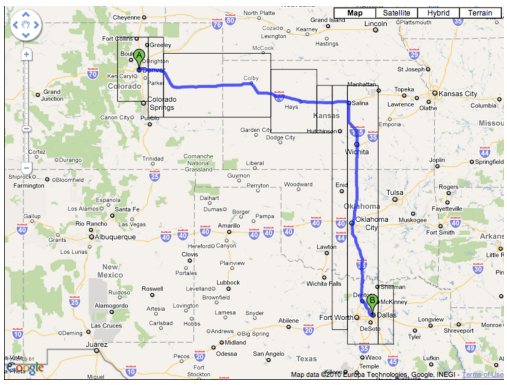


Σχήμα 13: RouteBoxer - Συγχώνευση πλατιών παραλληλογράμμων [12]

7. Σε μια δεύτερη προσέγγιση, ακολουθείται η ίδια τακτική, αλλά συγχωνεύονται τα κελιά που είναι **κάθετα γειτονικά** με άλλα κελιά, δημιουργώντας ένα σύνολο r_2 από ψηλά παραλληλόγραμμα, κάθε ένα πλάτους d.

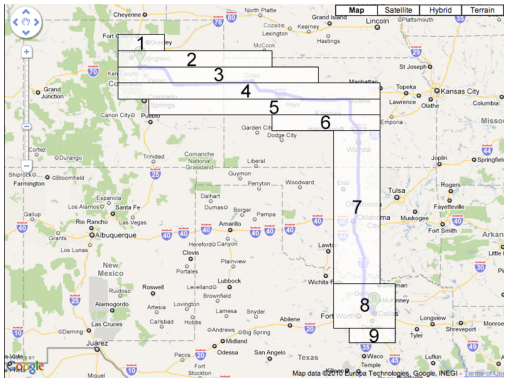


Σχήμα 14: RouteBoxer - Δημιουργία ψηλών παραλληλογράμμων [12]

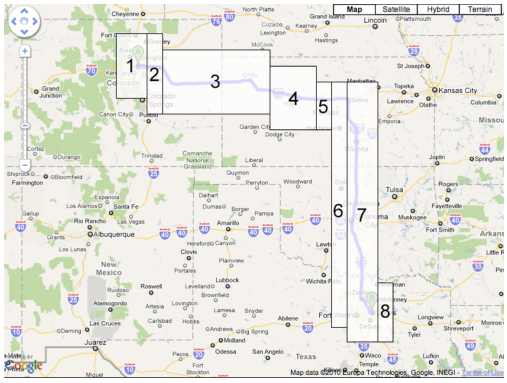


Σχήμα 15: RouteBoxer - Συγχώνευση ψηλών παραλληλογράμμων [12]

8. Ο αλγόριθμος επιστρέφει το σύνολο r με το μικρότερο πληθάριθμο, δηλαδή το $r = min(card(r_1), card(r_2))$.



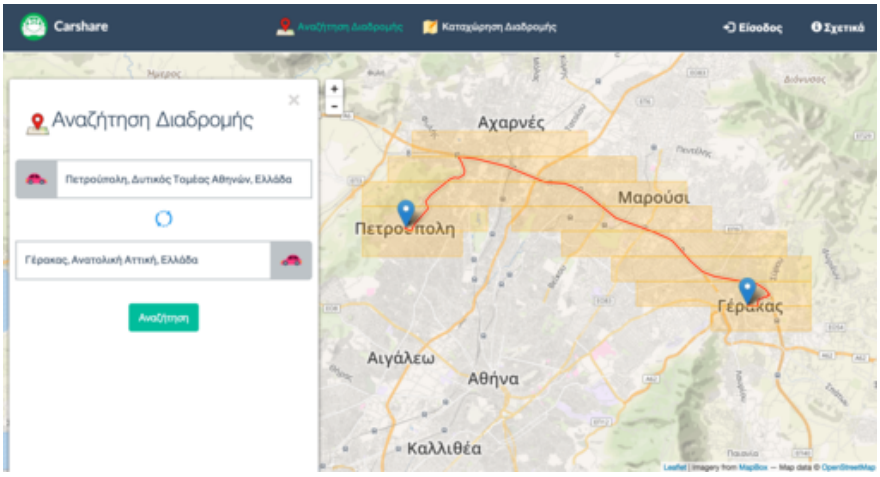
Σχήμα 16: RouteBoxer - Πλήθος πλατιών παραλληλογράμμων



Σχήμα 17: RouteBoxer - Πλήθος ψηλών παραλληλογράμμων

Εδώ αξίζει να σημειωθεί, ότι ο καθορισμός της σταθεράς d, εξαρτάται και πάλι από το μήκος της διαδρομής, με τρόπο ανάλογο του καθορισμού της σταθεράς ε του *LCSS*.

Εφαρμόζοντας την κλάση RouteBoxer για την διαδρομή που αναπαριστά η εικόνα 7, παίρνουμε το ακόλουθο σύνολο παραλληλογράμμων:

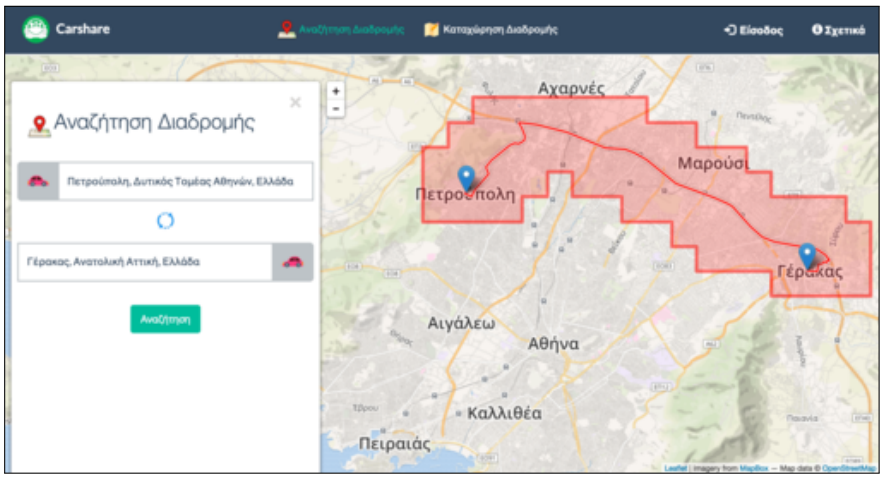


Σχήμα 18: Σύνολο παραλληλογράμμων για την διαδρομή Πετρούπολη - Γέρακας

3.4.2 Αλγόριθμος μετατροπής συνόλου παραλληλογράμμων σε πολύγωνο

Τελευταίο βήμα, είναι η μετατροπή του συνόλου των γειτονικών παραλληλογράμμων, σε ένα πολύγωνο, προκειμένου να σχηματισθεί το κατάλληλο *GeoJSON Πολύγωνο* που θα εφαρμοσθεί στον τελεστή *$geoIntersects*. Αυτό αποτελεί πρόβλημα υπολογιστικής γεωμετρίας και έχει προταθεί αλγόριθμος [13], όπου δεδομένου συνόλου σημείων, προκύπτει πάντα ένα μοναδικό πολύγωνο. Επειδή ο συγκεκριμένος αλγόριθμος δεν δέχεται κοινές κορυφές-σημεία, τον τροποποιούμε κατάλληλα, ώστε να απαλοίφουμε τις κοινές κορυφές.

Εφαρμόζοντας τον αλγόριθμο για την 18, παίρνουμε το πολύγωνο:



Σχήμα 19: Το πολύγωνο για την διαδρομή Πετρούπολη - Γέρακας

Έτσι με την χρήση του τελεστή *$geoIntersects*, θα αποκοπούν όλες οι διαδρομές που δεν τέμνονται με το *GeoJSON Πολύγωνο* μειώνοντας το πλήθος των διαδρομών προς αναζήτηση ομοιότητας. Είναι προφανές ότι σχεδόν όλες οι διαδρομές που αποκόπτονται, θα παρουσιάζαν μηδενική ή πάρα πολύ μικρή ομοιότητα, σε σχέση με την διαδρομή που αναζήτησε ο χρήστης.

4. ΔΟΜΗ ΚΑΙ ΤΕΧΝΙΚΕΣ ΥΛΟΠΟΙΗΣΗΣ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

4.1 Ακολουθώντας το Αρχιτεκτονικό Μοντέλο Ανάπτυξης REST

Το CarShare ακολουθεί το Αρχιτεκτονικό Μοντέλο Ανάπτυξης *REST*. Η επιλογή αυτή έγινε, ώστε να δημιουργηθεί μια καλά δομημένη εφαρμογή, ξεχωρίζοντας τα διαφορετικά κομμάτια ανάπτυξης λογισμικού μεταξύ τους και να τεθούν θεμέλια για μια εύκολη κλιμάκωση σε περαιτέρω μελλοντική ανάπτυξη της.

Το *REST*, αποτελεί πλέον ένα από τα πιο ενδεδειγμένα μοντέλα ανάπτυξης κατανεμημένων, διαδικτυακών υπηρεσιών (*web services*), όπου πρακτικά εφαρμόζονται κάποιοι συγκεκριμένοι περιορισμοί κατά την ανάπτυξη της εφαρμογής. Πιο συγκεκριμένα, για να χαρακτηριστεί μια διαδικτυακή υπηρεσία ως *RESTful* - δηλαδή διακτυακή υπηρεσία που ακολουθεί το μοντέλο *REST* - πρέπει να πληρούνται οι ακόλουθοι περιορισμοί [14] :

1. **Client-Server**: Υπάρχει μια κοινή διεπαφή, που επιτρέπει την ανεξαρτησία μεταξύ *πελάτη* (*client*) και εξυπηρετητή (*server*). Η ανεξαρτησία αυτή, επιτρέπει στον πελάτη να μην απασχολείται με λειτουργίες όπως η αποθήκευση δεδομένων, η οποία αποκρύπτεται από τους εξυπηρετητές, επιτρέποντας την μεταφερσιμότητα του κώδικα των πελατών. Παράλληλα, οι εξυπηρετητές δεν απασχολούνται καθόλου με λειτουργίες όπως η παρουσίαση των δεδομένων, μέσα από *Σελίδες Γλώσσας Σήμανσης Υπερκειμένου* (*HyperText Markup Language(HTML) Pages*) ή γενικότερες εργασίες που αφορούν το *frontend* της εφαρμογής.
2. **Stateless**: Κάθε αίτημα του πελάτη περιέχει όλην την απαραίτητη πληροφορία για την εξυπηρέτηση του. Ο εξυπηρετητής δεν αποθηκεύει καμία πληροφορία για την κατάσταση σύνδεσης του πελάτη, όπως οι συνεδρίες (sessions), αλλά αντί αυτού, όλη η απαραίτητη πληροφορία είναι αποθηκευμένη στον πελάτη. Η κατάσταση σύνδεσης μεταφέρεται σε κάθε αίτημα του πελάτη, στον εξυπηρετητή, για δραστηριότητες που αφορούν την σύνδεση με την βάση δεδομένων, την ταυτοποίηση του χρήστη και άλλες παρεμφερείς ενέργειες.
3. **Cacheable**: Κάθε πληροφορία του διαδικτύου μπορεί να αποθηκευτεί προσωρινά. Οι απαντήσεις θα πρέπει να μπορούν να ορίζουν ότι είτε επιτρέπουν να γίνονται cache, είτε όχι έτσι ώστε να αποτρέπεται η χρήση μη έγκυρης πληροφορίας. Καλά-ορισμένο μερικό ή πλήρες caching μπορεί βελτιώσει σημαντικά την κλιμάκωση και την απόδοση του συστήματος.
4. **Layered System**: Ο πελάτης δεν μπορεί να διακρίνει εάν είναι απευθείας συνδεδεμένος με τον τελικό εξυπηρετητή ή με άλλους ενδιάμεσους. Η χρήση ενδιάμεσων servers επιφέρει σημαντική βελτίωση κλιμάκωση και την απόδοση του συστήματος επιτρέποντας *εξισορρόπηση φόρτου* (*load balancing*) και διαμοιρασμένες (*shared*) caches.
5. **Code on demand (προαιρετικό)**: Είναι ο μοναδικός προαιρετικός περιορισμός του μοντέλου *REST*. Με αυτόν τον περιορισμό, ο πελάτης μπορεί να κατεβάσει και να

εκτελέσει κώδικα, ο οποίος παρέχεται από τον εξυπηρετητή (όπως Java applets, Javascript αρχεία, και άλλα). Βέβαια στην περίπτωσή μας, που εστιάζουμε στην δημιουργία ενός *REST API*, αυτός ο περιορισμός δεν είναι αναγκαίος, καθώς ο πελάτης απλά λαμβάνει μια απάντηση από έναν εξυπηρετητή και την επεξεργάζεται κατάλληλα. Αυτός ο περιορισμός αναφέρεται κυρίως σε περιστάσεις όπου θέλουμε να αναπτύξουμε κάποιον *περιηγητή* (*browser*) ή κάποιον *εξυπηρετητή διαδικτύου* (*web server*).

6. **Uniform interface (Κοινή Διεπαφή)**: Αποτελεί τον πιο σημαντικό περιορισμό για την κατασκευή μιας υπηρεσίας που ακολουθεί *REST*. Με αυτόν, απλοποιείται η ανάπτυξη της εφαρμογής, αφού απλοποιεί και αποσυνδέει τα συστατικά της αρχιτεκτονικής του συστήματος, επιτρέποντας κάθε ένα από αυτά να αναπτυχθεί αυτόνομα.

4.2 Παραλλαγή του MVC με χρήση Laravel και AngularJS

Προκειμένου να τηρηθούν όλοι οι περιορισμοί που θέτει το μοντέλο *REST*, έπρεπε να επιλεχθούν οι κατάλληλες τεχνολογίες. Αρχικά, υιοθετήθηκε το *MVC* για τον σχεδιασμό της *διεπαφής χρήστη* (*user interface*), προκειμένου να απλοποιήσουμε τα συστατικά της αρχιτεκτονικής του συστήματος (**Uniform interface**) [15].

Περιληπτικά, το *MVC*, αποτελείται από τρία συστατικά στοιχεία:

1. **Model**: Το *Μοντέλο* (*Model*), διαχειρίζεται τα δεδομένα, την λογική και τους κανόνες της εφαρμογής.
2. **View**: Η *Όψη* (*View*), μπορεί να είναι οποιοδήποτε είδος εξόδου που χρησιμοποιείται για παρουσίαση πληροφορίας, όπως για παράδειγμα είναι ένας κώδικας *HTML*.
3. **Controller**: Ο *Ελεγκτής* (*Controller*), μετατρέπει τα δεδομένα εισόδου σε εντολές για το Μοντέλο ή τον Ελεγκτή.

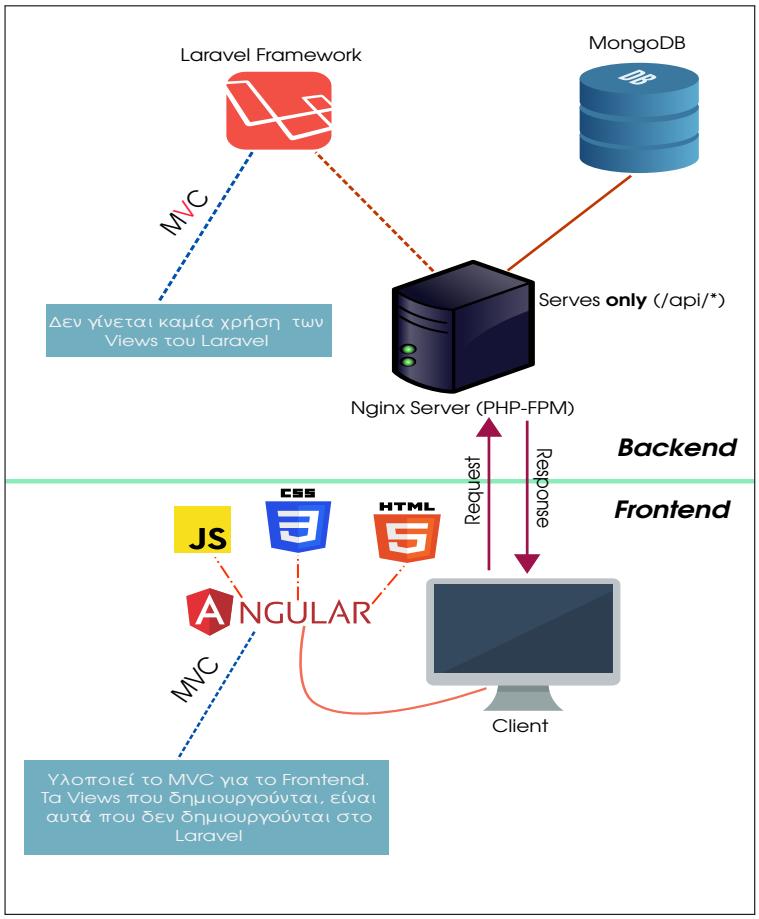
Το MVC, μπορεί να υπάρξει τόσο στο *Backend*, όσο και στο *Frontend* μιας εφαρμογής. Δεν υπάρχει κάποιος διαχωρισμός στην λογική που εφαρμόζεται το *MVC*, μεταξύ του αν αυτό εφαρμόζεται στο *Backend* ή στο *Frontend*, το μόνο που αλλάζει είναι τα *συστατικά* (*components*) που αυτό διαχείριζεται. Για το *Frontend*, τα συστατικά είναι όλα τα κομμάτια που είναι “αόρατα” στον πελάτη, όπως η *HTML*, *CSS*, αρχεία *Javascript*, ενώ για το *Backend*, τα συστατικά είναι ο *κώδικας παρασκηνίου* (*serverside code*), ανάλογα με την γλώσσας προγραμματισμού που έχει επιλεχθεί (*PHP, Ruby, Java,...*).

Στο *Backend* της εφαρμογής το *Μοντέλο*, συνήθως ορίζει οντότητες της εφαρμογής, οι οποίες αλληλεπιδρούν με την βάση δεδομένων με συγκεκριμένες ενέργειες. Η *Όψη*, μπορεί να είναι μια δυναμική σελίδα *HTML* που δημιουργείται από συγκεκριμένα δεδομένα, μια *JSON* απάντηση ή οποιαδήποτε άλλη έξοδος επιλέξει να δείξει ο εξυπηρετητής στον πελάτη. Οι *Ελεγκτές*, ουσιαστικά “μεταφράζουν” έναν *σύνδεσμο* (*url*), σε κάποια πράξη της εφαρμογής.

Όσον αφορά το *Frontend*, το *Μοντέλο*, συνήθως ορίζει οντότητες που έχουν κάποια κοινή λειτουργία μεταξύ τους. Η *Όψη* επιτρέπει να αναλυθεί μια σύνθετη αναπάρασταση, όπως για παράδειγμα μια σελίδα *HTML*, σε απλούστερα κομμάτια που ενώνονται μεταξύ τους, αυξάνοντας έτσι την εκφραστικότητα και την αφαιρετικότητα της εφαρμογής. Τέλος, οι *Ελεγκτές*, δέχονται *γεγονότα* (*events*), που συμβαίνουν σε κάποιο κομμάτι της *Όψης* και τα "μεταφράζουν" σε ενέργειες.

Στην εφαρμογή μας, επιλέχθηκε σαν γλώσσα προγραμματισμού για τον εξυπηρετητή, η *PHP* [16]. Υπάρχουν πολλά *Frameworks*,που χρησιμοποιούνται στην *PHP* για το *MVC*, και αυτό που επιλέχθηκε στο CarShare, είναι το *Laravel* [17]. Στο *Frontend*, χρησιμοποιήθηκε το *AngularJS Framework*[18].

Στην προσπάθεια, όμως να συνδυαστούν δύο *MVC Frameworks*, ένα για το *Backend* και ένα για το *Frontend*, είναι φανερό ότι η *Όψη* πρέπει να υλοποιηθεί σε ένα από τα δύο. Αν ήταν το *Backend*, αυτό που υλοποιεί την *Όψη*, τότε θα παραβιάζαμε τον περιορισμό **Client-Server**, καθώς ο εξυπηρετητής θα απασχολούταν με την παρουσίαση των δεδομένων. Επομένως, αυτό που ακολουθήθηκε, είναι η *Όψη* να υλοποιείται μόνο από την *AngularJS*. Οποιαδήποτε δυναμική πληροφορία θέλει να λάβει ο πελάτης, επικοινωνεί μέσω του **Uniform Interface** με τον εξυπηρετητή. Συνοψίζοντας όλα αυτά που αναφέρθηκαν, η αρχιτεκτονική του CarShare, σχηματικά είναι η ακόλουθη:

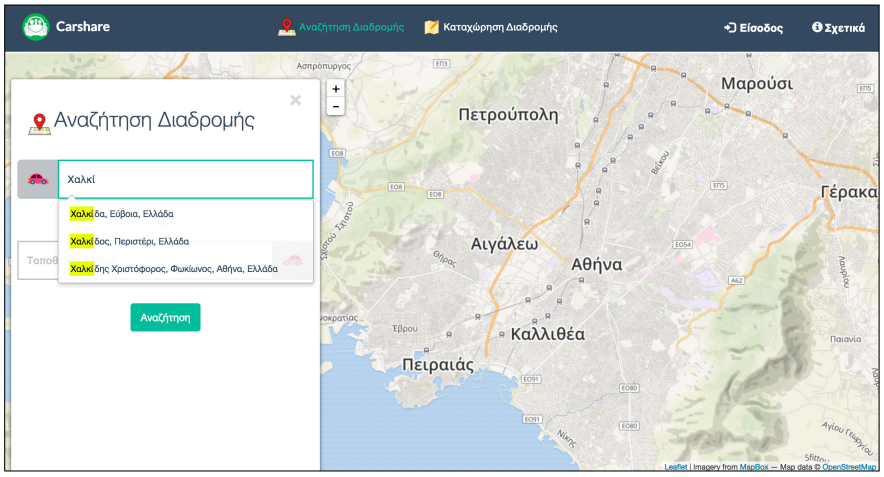


Σχήμα 20: Η αρχιτεκτονική του CarShare

4.3 Χρήση του Google Geocoding και του Leaflet στο CarShare

Το CarShare, χρησιμοποιεί το *Leaflet.js* [19]. Το *Leaflet.js*, είναι μια βιβλιοθήκη *Javascript*, που παρέχει διαδραστικούς χάρτες, προσφέροντας πάρα πολλές δυνατότητες στον προγραμματιστή.

Στην εφαρμογή, ο χρήστης κατά την πληκτρολόγηση της τοποθεσίας προορισμού ή άφιξης, λαμβάνει ένα πλήθος από διαθέσιμες τοποθεσίες που μπορεί να επιλέξει, όπως φαίνεται παράκατω :



Σχήμα 21: Προτάσεις Τοποθέσιας Αναχώρησης

Αυτό επιτυγχάνεται με την χρήση του *Google Maps API* [20]. Το API, με βάση την τοποθεσία που πληκτρολογεί ο χρήστης, προσπαθεί να μαντέψει ολόκληρο το όνομα της τοποθεσίας. Επιπλέον, το API χρησιμοποιείται για να βρεθούν οι συντεταγμένες της τοποθεσίας με βάση το όνομα της. Αυτή η διαδικασία είναι γνωστή ως *Geocoding*.

Στην συνέχεια, η διαδρομή που πρόκειται να κάνει ο οδηγός ή η διαδρομή που αναζητεί ο χρήστης-συνεπιβάτης, βρίσκεται με την χρήση του *Leaflet Routing Machine* [21]. Το συγκεκριμένο λογισμικό, προσφέρει στον προγραμματιστή την δυνατότητα, να υπολογίσει την πιο γρήγορη διαδρομή μεταξύ δύο σημείων, γεγονός που το καθιστά ιδανικό για το CarShare.

4.4 Ταυτοποίηση χρηστών

Σε αυτό το κομμάτι της πτυχιακής, αξίζει να αναφερθεί, ότι δώθηκε ιδιαίτερη σημασία στον τρόπο που θα γίνεται η ταυτοποίηση των χρηστών. Όπως αναφέρθηκε στην ενότητα 2.2, η είσοδος στο CarShare γίνεται μέσω του *Facebook* και αυτό θα αποτελέσει το μόνο μέσο για την ταυτοποίηση. Μια παραδοσιακή πρακτική, θα ήταν η αποθήκευση των στοιχείων του χρήστη που λαμβάνονται από το *Facebook Graph API* [22] σε κάποια PHP-συνεδρία (session). Κάτι τέτοιο όμως δεν ενδείκνυται για την δημιουργία ενός *RESTful API*, όπως το *CarShare*, καθώς ο εξυπηρετητής αποθηκεύει δεδομένα που θα μπορούσαν να αποθηκευτούν και στον πελάτη, αυξάνοντας έτσι τον φόρτο του. Βέβαια, το να αποθηκεύο-

νται δεδομένα που αφορούν την ταυτοποίηση των χρηστών στον πελάτη, ίσως εγείρουν ανησυχίες σχετικά με την ασφάλεια του συστήματος. Για αυτό, χρησιμοποιήθηκαν οι Διαδικτυακές Λεκτικές Μονάδες JSON (*JWT* στο εξής).

4.4.1 Facebook Graph API

Προκειμένου ο χρήστης να πραγματοποιήσει κάποιες ενέργειες, όπως είναι η Καταχώρηση Διαδρομής (2.3) ή η Προσθήκη ως Συνεπιβάτης (2.4), απαραίτητο είναι να έχει συνδεθεί μέσω του λογαριασμού του στο *Facebook*. Στην *Laravel*, χρησιμοποιήσαμε το λογισμικό *Socialite* [23], το οποία προσφέρει μια εύκολη και εκφραστική διεπαφή για την *OAuth 2.0 ταυτοποίηση* [24] του *Facebook*, καθώς και άλλων γνωστών μέσων κοινωνικής δικτύωσης. Το μοναδικό δεδομένο που χρειάζεται το CarShare από το API, είναι το αναγνωριστικό (*identifier*) του χρήστη στο *Facebook*.

Γνωρίζοντας το αναγνωριστικό του χρήστη, χρησιμοποιούμε το *Facebook Graph API*, στην πλευρά του εξυπηρετητή για να λάβουμε επιπλέον δεδομένα για προβολή, όπως το όνομά του, το επώνυμό του και άλλα.

4.4.2 JSON Web Tokens (JWT)

Σήμερα, αρκετά συχνό φαίνομενο, είναι οι εξυπηρετητές να καθυστερούν να απαντήσουν στα αιτήματα των πελατών, καθώς πολύ μεγάλος αριθμός πελατών έχει συνδεθεί σε αυτούς. Η πιο κλασσική προσέγγιση, ήταν σε μια βάση δεδομένων να υπάρχει για κάθε χρήστη μια εγγραφή με την διεύθυνση του ηλεκτρονικού ταχυδρομείου (*email*) και ο κατακερματισμένος του κωδικός. Έτσι, κατά την είσοδο στο σύστημα, γινόταν ερώτηση στην βάση δεδομένων για να ελεγχθεί αν το *email* και ο κωδικός είναι σωστά και σε περίπτωση επιτυχίας, δημιουργούταν στον εξυπηρετητή ένα αρχείο-session (συνήθως στον κατάλογο ***/*etc** για την *PHP*), όπου αντιστοιχούσε σε μια ενεργή σύνδεση του πελάτη στον server.

Είναι εύκολα αντιληπτό, ότι σε εφαρμογές που εξυπηρετούν εκατομμύρια πελάτες, θα πρέπει :

1. Να διατηρούν εκατομμύρια εγγραφές στην βάση δεδομένων με το *email* και τον κωδικό, μια για κάθε λογαριασμό του κάθε πελάτη και ακόμα περισσότερο,
2. Να διατηρούν εκατομμύρια αρχεία-*session* που θα αυξάνουν δραματικά τον χρόνο απόκρισης του εξυπηρετητή.

Το πρώτο πρόβλημα, προφανώς επιλύθηκε, αφού το *Facebook* είναι αυτό που διατηρεί το αρχείο για τους χρήστες του και κατά επέκταση και για τους δικούς μας. Στην εφαρμογή μας το δεύτερο πρόβλημα, αντιμετωπίστηκε με την χρήση των *JWT*. Η κεντρική ιδέα, είναι η σύνδεση του χρήστη, να αποθηκεύεται στον πελάτη και όχι στον εξυπηρετητή.

Τα *JWT* [25], αποτελούν ένα πρότυπο το οποίο περιγράφει έναν ασφαλή τρόπο μετάδοσης πληροφοριών, αναπαριστόντας αυτές, σαν ένα *JSON* αντικείμενο. Οι πληροφορίες μπορούν να πιστοποιηθούν και να τις εμπιστευτούμε, διότι υπογράφονται ψηφιακά. Υπο-

γράφονται χρησιμοποιώντας ένα “μυστικό” (με την χρήση του αλγόριθμου *HMAC* [26]) ή με την χρήση δημόσιου/ιδιωτικού κλειδιού με *RSA* [27].

Τα *JWT*, αποτελούνται από τρία JSON, κωδικοποιημένα σε συμβολοσειρές με *base64* [28] , τα οποία μεταξύ τους διαφοροποιούνται με την τελεία (.) :

1. Η επικεφαλίδα (*header*), η οποία περιλαμβάνει το πεδίο **alg** και το **typ**, όπου αντίστοιχα ορίζεται ο αλγόριθμος κατακερματισμού (είτε **HMAC SHA256** ή **RSA**) και ο τύπος όπου πάντα ορίζεται σε **JWT**.
2. Το δεύτερο μέρος ονομάζεται *payload* και περιλαμβάνει όλες τις πληροφορίες που θέλουμε εμείς να αποθηκεύσουμε, καθώς και άλλες χρήσιμες πληροφορίες, όπως το **exp** που φέρει τον χρόνο λήξης του *JWT*, το **aud** στο οποίο ορίζουμε κατηγορίες χρηστών (ιδιαίτερα χρήσιμο σε εφαρμογές που θέλουμε να δώσουμε διαφορετικά δικαιώματα στους χρήστες ανάλογα στην ομάδα που ανήκουν), καθώς και άλλα.
3. Το τρίτο και τελευταίο μέρος, είναι η υπογραφή (*signature*). Για να δημιουργηθεί αυτό το τμήμα, χρησιμοποιείται το κωδικοποιημένο *header*, το κωδικοποιημένο *payload*, το μυστικό και ο αλγόριθμος του *header*. Η υπογραφή, χρησιμοποιείται για να πιστοποιηθεί, ότι ο αποστολέας είναι πράγματι αυτός που λέει ότι είναι και ότι το μήνυμα δεν αλλοιώθηκε μέχρι να έρθει στον εξυπηρετητή.

Στην πλευρά του εξυπηρετητή το μόνο που χρειαζόμαστε είναι ένα και μοναδικό κλεδί, του οποίου οι χρήσεις είναι να δημιουργεί τα JWT και να τα πιστοποιεί. Χρησιμοποιήθηκε το λογισμικό *Laravel: jwt-auth* [29], το οποίο απλοποιεί την διαδικασία της ταυτοποίησης και δημιουργίας των *JWT* στο *Laravel Framework*. Σε αυτό το σημείο, πρέπει να αναφέρουμε ότι ο πελάτης, είναι αναγκαίο να στέλνει το *JWT* του, ακολουθώντας το *Bearer schema* [30]. Σε αυτό το σχήμα, στέλνεται το *JWT* σαν επικεφαλίδα με την ακόλουθη μορφή:

```
Authorization: Bearer <jwt_token>
```

Στο CarShare, χρειάζεται να υπάρχει αποθηκευμένος στον πελάτη μόνο ο *identifier* του στο *Facebook* (το **fb_id**) και έτσι ένα τυπικό παράδειγμα *JWT*, είναι το ακόλουθο:

Σχήμα 22: Ένα παράδειγμα JWT στο CarShare

4.5 Υλοποίηση του αλγορίθμου *LCSS* σε *PHP-CPP*

Σε μια πρώτη προσέγγιση, ο αλγόριθμος *LCSS*, αναπτύχθηκε σε “καθαρή” PHP. Παρατηρήθηκε, ότι η υλοποίηση δεν ήταν καθόλου αποδοτική, ιδιαίτερα όσο αύξανε το μήκος της διαδρομής προς αναζήτηση. Σε στάδιο *profiling*, ανακαλύφθηκε ότι σχεδόν όλος ο χρόνος στην εκτέλεση του *LCSS* αφιερωνόταν στον πίνακα πρωτόγονου τύπου **array** του δυναμικού προγραμματισμού, που χρησιμοποιούμε για τον αλγόριθμο.

Σε μια δεύτερη προσέγγιση, δηλώσαμε τον πίνακα αντί για τον τύπο **array**, ως αντικείμενο κλάσης **SplFixedArray** [31]. Αυτή η κλάση, υπόσχεται ταχύτερη υλοποίηση του πίνακα, σε σχέση με τον κανονικό τύπο **array**, εφόσον γνωρίζουμε εξ’αρχής το μέγεθος του πίνακα, πράγμα το οποίο συμβαίνει και στην περίπτωση του *LCSS*. Παρόλα αυτά, η επιτάχυνση του προγράμματος, δεν ήταν ικανοποιητική. Γενικότερα, διαπιστώθηκε ότι η *PHP*, δεν ενδείκνυται για υλοποίηση αλγορίθμων που σχετίζονται με διαχείριση πινάκων, ειδικά αυτών που είναι πάνω από δύο διαστάσεων. Αυτό έγκειται στο γεγονός, ότι η υλοποίηση του πίνακα στο χαμηλό επίπεδο της PHP, υλοποιείται σαν ένας πίνακας κατακερματισμού με συνδεδεμένη λίστα, χάνοντας την άμεση πρόσβαση που έχουμε στην μνήμη σε μια γλώσσα που μεταγλωττίζεται πριν την εκτέλεση.

Για αυτό, ο *LCSS* υλοποιήθηκε σαν προέκταση (*extension*) της *PHP* σε *C++* με την χρήση της βιβλιοθήκης **PHP-CPP** [32]. Αυτή η βιβλιοθήκη επιτρέπει την υλοποίηση *PHP extensions*, χωρίς να απασχοληθεί ο προγραμματιστής με τα δύστροπα σημεία του *Zend Engine* [33] και την διαχείριση δεικτών, που αυτός απαιτεί.

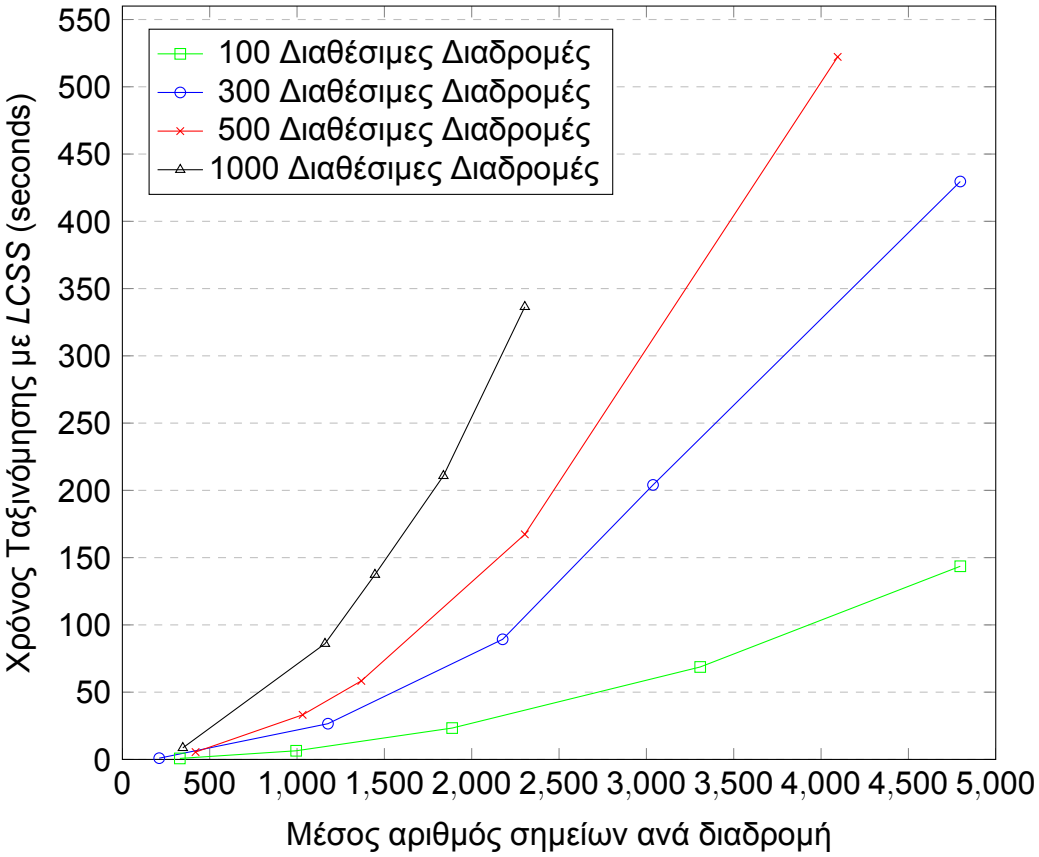
Όσον αφορά στην απόδοση του *LCSS*, η διαφορά που παρατηρήθηκε σε σχέση με τις προηγούμενες προσεγγίσεις, ήταν πολύ μεγαλή. Ο κώδικας του *LCSS* που αναπτύχθηκε στα πλαίσια αυτής της πτυχιακής, παραχέται σαν κώδικας ανοικτού λογισμικού [34].

4.6 Benchmarking

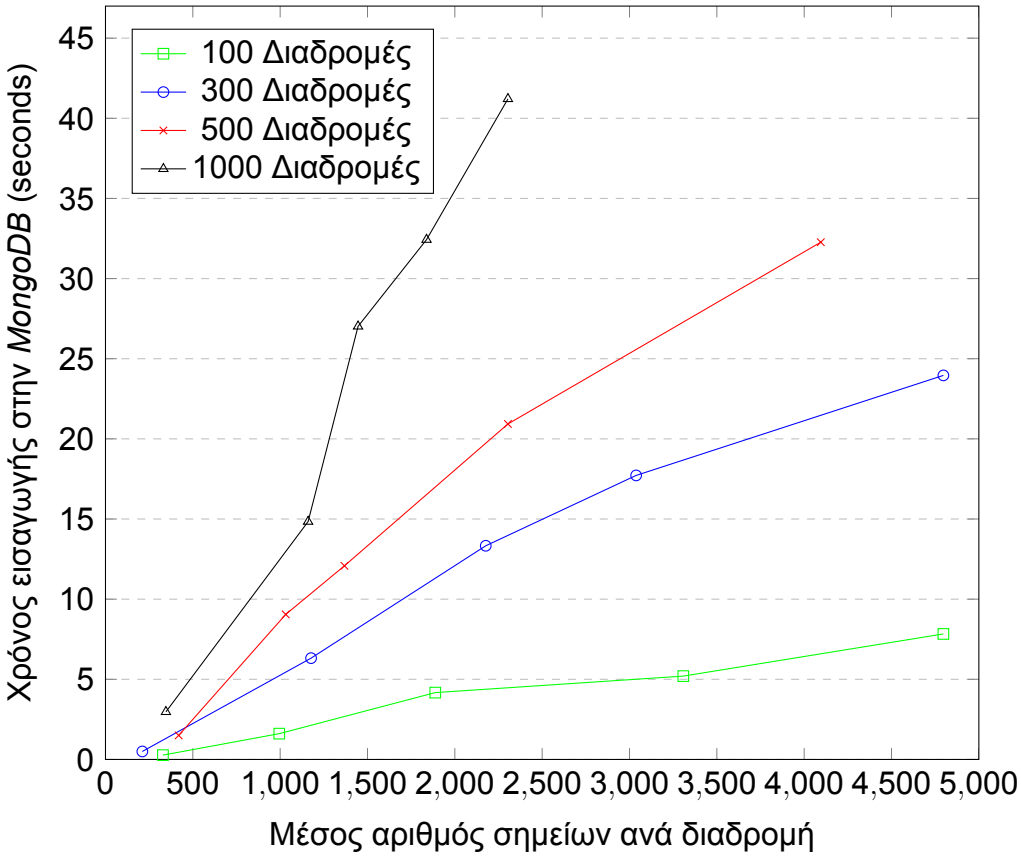
Σε αυτή την ενότητα, παρουσιάζεται η κλιμάκωση του *LCSS*, όσο αυξάνονται οι υποψήφιες διαδρομές στην βάση δεδομένων, σε σχέση με την διαδρομή που αναζητεί ο χρήστης, καθώς και ο χρόνος που κάνει η *MongoDB* να εισάγει μαζικά διαδρομές στην βάση δεδομένων. Οι χρόνοι για τον *LCSS*, αναφέρονται αποκλειστικά στην υλοποίηση σε *C++* που αναφέραμε προηγουμένως και δεν συμπεριλαμβάνεται ο πολύ μικρός χρόνος που κάνει η *MongoDB* να αποκόψει τίς διαδρομές. Όλοι οι παρακάτω χρόνοι αναφοράς, είναι σε δευτερόλεπτα.

Αποτελέσματα Benchmarking			
Μέσος αριθμός σημείων διαδρομής	Πλήθος Διαδρομών	Χρόνος Εισαγωγής στην MongoDB	Χρόνος Εκτέλεσης LCSS
329	100	0.27	0.74
995	100	1.61	6.43
1888	100	3.17	23.32
3308	100	5.20	68.72
4797	100	7.83	143.59
211	300	0.4987	0.93
1177	300	6.32	26.56
2177	300	13.33	89.28
3038	300	17.72	204.06
4797	300	23.96	429.59
419	500	1.50	5.48
1032	500	9.05	33.15
1368	500	12.08	58.38
2304	500	20.93	167.32
4095	500	32.27	522.21
346	1000	2.97	8.59
1160	1000	14.84	86.02
1446	1000	27.02	137.35
1838	1000	32.43	210.79
2304	1000	41.21	336.44

Είναι πολύ σημαντικό να κατανοηθεί, ότι στην πράξη δύσκολα θα υπάρξει τόσο μεγάλος αριθμός διαθέσιμων διαδρομών προς εύρεση ομοιότητας. Για παράδειγμα, η τελευταία γραμμή του πίνακα υποδηλώνει, ότι υπάρχουν 1000 οδηγοί που έχουν καταχωρήσει από μια διαδρομή της τάξης "*Αθήνα - Θεσσαλονίκη*", εκ των οποίων διαδρομών όλες πέρασαν σαν υποψήφιες από τον αλγόριθμο *RouteBoxer*. Αξίζει να αναφερθεί, ότι οι χρήστες-οδηγοί της εφαρμογής, μπορεί να δράσουν και σαν συνεπιβάτες, βλέποντας ότι ήδη υπάρχουν διαδρομές με μεγάλη ομοιότητα και να μην υποβάλλουν την διαδρομή τους απλά και μόνο για να είναι αυτοί οι οδηγοί. Παρόλα αυτά, δεν μπορεί να αποκλειστεί το ενδεχόμενο να υπάρχουν σε ένα σύστημα, τόσες πολλές και τόσο μεγάλες διαθέσιμες διαδρομές.



Σχήμα 23: Benchmarking της απόδοσης του LCSS



Σχήμα 24: Benchmarking της εισαγωγής των διαδρομών στην MongoDB

5. ΣΥΜΠΕΡΑΣΜΑΤΑ

Όπως φαίνεται από τα αποτελέσματα του *Benchmarking*, ο εξυπηρετητής αργεί να παρουσιάσει τα αποτελέσματα της αναζήτησης, όσο αυξάνουν το μήκος της διαδρομής και το πλήθος των διαθέσιμων διαδρομών. Όμως το κύριο πρόβλημα της εφαρμογής δεν είναι αυτό, αφού όπως ειπώθηκε, δεν είναι τόσο πιθανόν να συμβεί κάτι τέτοιο στην καθημερίνη χρήση.

Έμφαση πρέπει να δωθεί στον αλγόριθμο αποκοπής υποψήφιων διαδρομών από την βάση δεδομένων, ώστε να αποκόπτονται ακόμα περισσότερες διαδρομές που παρουσιάζουν μικρή ομοιότητα. Στην παρούσα φάση, οποιαδήποτε διαδρομή από την βάση δεδομένων, τέμνεται με το πολύγωνο που δημιουργείται από την διαδρομή αναζήτησης, θα υπολογιστεί η ομοιότητά της με τον *LCSS*. Αυτό σημαίνει ότι ακόμα και στα όρια του πολυγώνου να είναι η τομή τους, θα τρέξει ο αλγόριθμος *LCSS*, όπου θα υπολογιστεί η (μηδενική) ομοιότητα των δύο διαδρομών.

Ένας τρόπος για να αποκόπτουμε περισσότερες διαδρομές, θα ήταν οι διαδρομές που καταχωρούν οι οδηγοί, να αποθηκεύονται ως πολύγωνα (όπως γίνεται στην περίπτωση των διαδρομών προς αναζήτησης). Στην συνέχεια, θα ήταν δυνατό να υπολογιστεί το ποσοστό επικάλυψης των δύο διαδρομών, υπολογίζοντας τον λόγο των εμβαδών των δύο πολυγώνων. Αν αυτός ο λόγος, ήταν μεγαλύτερος από μια σταθερά-κατώφλι, τότε η διαδρομή θα συμπεριλαμβανόταν στον *LCSS*.

ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

Δυναμική Χρονοστρέβλωση	Dynamic Time Warping
Στιγμιότυπο Οθόνης	Screenshot
Μη Σχεσιακή Βάση Δεδομένων	NoSQL Database
Παρασκήνιο	Backend
Προσκήνιο	Frontend
Σκελετός	Framework
Μοντέλο-Όψη-Ελεγκτή	Model-View-Controller
Ταυτοποίηση	Authentication
Ταυτοποίηση Βασισμένη σε Λεκτικές Μονάδες	Token-based Authentication
Διαδικτυακών Λεκτικών Μονάδων JSON	JSON Web Tokens
Συνεδρία	Session
Κέντρα Δεδομένων	Data Centers
Τομή	Intersection
Συμπερίληψη	Inclusion
Εγγύτητα	Proximity
Διαδικτυακή Υπηρεσία	Web Service
Σελίδες Γλώσσας Σήμανσης Υπερκειμένου	HyperText Markup Language Pages
Εξισορρόπηση Φόρτου	Load Balancing
Περιηγητής	Browser
Εξυπηρετητής Διαδικτύου	Web Server
Διεπαφή Χρήστη	User Interface
Σύνδεσμος	Url
Συστατικά	Components
Κώδικας Παρασκηνίου	Serverside Code
Κοινή Διεπαφή	Uniform Interface
Αναγνωριστικό	Identifier

ΣΥΝΤΜΗΣΕΙΣ, ΑΡΚΤΙΚΟΛΕΞΑ ΚΑΙ ΑΚΡΩΝΥΜΙΑ

DTW	Dynamic Time Warping
LCSS	Longest Common Subsequence
REST	Representational State Transfer
MVC	Model-View-Controller
HTML	HyperText Markup Language Pages

ΑΝΑΦΟΡΕΣ

[1] Jonker, R.; De Leve, G.; Van Der Velde, J. A.; Volgenant, A. (May 1980), ”Rounding symmetric traveling salesman problems with an asymmetric assignment problem”, Operations Research pp. 623–627.

[2] Donald J. Berndt, James Clifford ”Using Dynamic Time Warping to Find Patterns in Time Series” *AAAI Technical Report WS-94-03, 1994*

[3] Michail Vlachos, George Kollios, Dimitrios Gunopulos ”Discovering Similar Multidimensional Trajectories”, *IEEE Computer Society Washington, DC, USA 2002.*

[4] Chotirat Ann Ratanamahatana, Eamonn Keogh ”Three Myths about Dynamic Time Warping Data Mining”, Department of Computer Science and Engineering, University of California, Riverside, CA 92521

[5] The MongoDB 3.2 Manual, **Available:** https://docs.mongodb.org/manual/

[6] Cornelia GYŐRÖDI, Robert GYŐRÖDI, George PECHERLE, Andrada OLAH ”A comparative study: MongoDB vs. MySQL” *Engineering of Modern Electric Systems (EMES), 2015 13th International Conference*

[7] MongoDB and 2dsphere Indexes, **Available:** https://docs.mongodb.com/manual/core/2dsphere/

[8] IETF Geographic JSON Working Group ”The GeoJSON Format Specification”, **Available:** http://geojson.org/geojson-spec.html

[9] MongoDB: Query Selectors, **Available:** https://docs.mongodb.com/master/reference/operator/query/

[10] IETF Geographic JSON Working Group ”LineString as GeoJson Object”, **Available:** http://geojson.org/geojson-spec.html#linestring

[11] The RouteBoxer class, **Available:** https://code.google.com/archive/p/routeboxer-java/

[12] RouteBoxer Algorithm: Images, **Available:** https://github.com/printercu/google-maps-utility-library-v3-read-only/tree/master/routeboxer/docs

[13] Joseph O’Rourke ”Uniqueness of Orthogonal Connect-the-Dots” *Elsevier Science Publishers B.V.(North-Holland), 1988*

[14] The Representational State Transfer wiki, **Available:** https://en.wikipedia.org/wiki/Representational_state_transfer

[15] The Model-View-Controller wiki, **Available:** https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

[16] PHP Manual, **Available:** http://php.net/manual/en/

[17] The Official Laravel Documentation, **Available:** https://laravel.com/docs

[18] AngularJS Documentation, **Available:** https://docs.angularjs.org/api

[19] Leaflet: an open-source JavaScript library for mobile-friendly interactive maps, **Available:** http://leafletjs.com/

[20] Google Maps APIs, Geocoding Documentation, **Available:** https://developers.google.com/maps/documentation/geocoding/intro

[21] Control for Routing in Leaflet (Documentation), **Available:** https://github.com/perliedman/leaflet-routing-machine

[22] Facebook: The Graph API, **Available:** https://developers.facebook.com/docs/graph-api

[23] Socialite on Github, **Available:** https://github.com/laravel/socialite

[24] D. Hardt, Ed.(Microsoft) ”The OAuth 2.0 Authorization Framework” *Internet Engineering Task Force (IETF), RFC 6749, October 2012*, **Available:** https://tools.ietf.org/html/rfc6749

[25] M.Jones(Microsoft), J.Bradley(Ping Identity), N.Sakimura(NRI) ”JSON Web Token (JWT)” *Internet Engineering Task Force (IETF), RFC 7519, May 2015*, **Available:** https://tools.ietf.org/html/rfc7519

[26] H.Krawczyk(IBM), M.Bellare(UCSD), R.Canetti(IBM) ”HMAC: Keyed-Hashing for Message Authentication” *Network Working Group, RFC 2104, February 1997*, **Available:** https://www.ietf.org/rfc/rfc2104.txt

[27] J.Jonsson, B.Kaliski (RSA Laboratories) ”Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1” *Network Working Group, RFC 3447, February 2003*, **Available:** https://www.ietf.org/rfc/rfc3447.txt

[28] S.Josefsson, Ed. ”The Base16, Base32, and Base64 Data Encodings” *Network Working Group, RFC 3548, July 2003*, **Available:** https://tools.ietf.org/html/rfc3548

[29] Sean Tymon JSON Web Token Authentication for Laravel & Lumen, **Available:** https://github.com/tymondesigns/jwt-auth

[30] M.Jones(Microsoft), D.Hardt(Independent) ”The OAuth 2.0 Authorization Framework: Bearer Token Usage” *Internet Engineering Task Force (IETF), RFC 6750, October 2012*, **Available:** https://tools.ietf.org/html/rfc6750

[31] PHP Manual: The SplFixedArray class, **Available:** http://php.net/manual/en/class.splfixedarray.php

[32] PHP-CPP: A C++ library for developing PHP extensions, **Available:** http://www.php-cpp.com/

[33] Zend Engine wiki, **Available:** https://en.wikipedia.org/wiki/Zend_Engine

[34] LCSS- An implementation of LCSS Algorithm written in php-cpp, **Available:** https://github.com/ThemisB/LCSS