ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΣΧΕΔΙΑΣΗ ΚΑΙ ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ ΑΝΑΖΗΤΗΣΗΣ ΚΑΙ ΔΙΑΧΕΙΡΙΣΗΣ ΕΣΤΙΑΤΟΡΙΩΝ

ΝΙΚΟΛΑΟΣ ΤΡΙΚΑΛΙΩΤΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Αντώνιος Δαδαλιάρης
Επίκουρος Καθηγητής

Λαμία 03/03/2022

UNIVERSITY OF
THESSALY

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE & TELECOMMUNICATIONS

DESIGN AND DEVELOPMENT OF RESTAURANT SEARCH AND MANAGEMENT APPLICATION

NIKOLAOS TRIKALIOTIS

FINAL THESIS

ADVISOR

Antonios Dadaliaris
Professor

Lamia 03/03/2022

«Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [(1)], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

*1. Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί **χωρίς να τα περικλείω σε εισαγωγικά** και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

*2. Δέχομαι ότι η αυτολεξεί **παράθεση χωρίς εισαγωγικά**, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια

4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.

Ημερομηνία: 03/03/2022

Ο Δηλ.
Νικόλαος Τρικαλιώτης

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση
του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων
σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.»

Περίληψη

Στην παρούσα πτυχιακή εργασία, σχεδιάστηκε και κατασκευάστηκε μια εφαρμογή αναζήτησης και διαχείρισης εστιατορίων σε μορφή ιστοσελίδας. Η ιστοσελίδα λειτουργεί σαν ένα ευρετήριο εστιατορίων, ένας ταξιδιωτικός οδηγός στοχευμένος στον επισιτιστικό κλάδο, στην οποία ο χρήστης μπορεί να αναζητήσει το εστιατόριο που ψάχνει με βάση το όνομά του, το φαγητό το οποίο σερβίρει ή την τοποθεσία στην οποία βρίσκεται και να μάθει περισσότερες πληροφορίες γι' αυτό. Μπορεί να διαβάσει κριτικές άλλων χρηστών, να αφήσει τη δική του ή ακόμα και να καταχωρήσει ένα καινούργιο εστιατόριο για να βοηθήσει μελλοντικούς χρήστες. Ο κώδικας για τη δομή της εφαρμογής γράφτηκε εξ' ολοκλήρου στο χέρι χωρίς τη βοήθεια κάποιου συστήματος διαχείρισης περιεχομένου (cms). Στόχος της εργασίας ήταν να ανακαλύψω πως δημιουργείται μια διαδικτυακή εφαρμογή τέτοιου τύπου από το μηδέν μέχρι και το στάδιο που βρίσκεται online και είναι προσβάσιμη από όλους.

Περιεχόμενα

1. Εισαγωγή στην ανάπτυξη ιστοσελίδων 6
   1.1 Ιστορία του Internet 6
   1.2 Internet και Ιστοσελίδες 7
   1.3 Front-end και back-end 8
2. HTML, CSS και Javascript 8
   2.1 Εισαγωγή 8
   2.2 CSS 10
   2.3 Javascript 11
3. Σχεδιασμός εφαρμογής 12
   3.1 Ανάλυση λειτουργιών 12
   3.2 Node.js και Express 14
   3.3 EJS 15
   3.4 Βάση δεδομένων: MongoDB 16
   3.5 Bootstrap 17
4. Ανάπτυξη της εφαρμογής 17
   4.1 Βασική λειτουργικότητα 17
   4.2 Στυλ εφαρμογής 22
   4.3 Διαχείριση σφαλμάτων 23
   4.4 Προσθήκη αξιολογήσεων 23
   4.5 Αυθεντικοποίηση χρηστών 25
      4.5.1 Συνεδρίες 25
      4.5.2 Passport 26
      4.5.3 Εξουσιοδότηση χρήστη 27
   4.6 Διαμόρφωση κώδικα 28
   4.7 Προσθήκη φωτογραφιών 28
   4.8 Προσθήκη χάρτη 30
   4.9 Επιπλέον στοιχεία 32
   4.10 Ασφάλεια 33
   4.11 Πηγαίνοντας online 34
5. Τελικό αποτέλεσμα 36
6. Συμπεράσματα και μελλοντικές επεκτάσεις 41

ΒΙΒΛΙΟΓΡΑΦΙΑ – ΔΙΑΔΙΚΤΥΑΚΟΙ ΤΟΠΟΙ 42

1. Εισαγωγή στην ανάπτυξη ιστοσελίδων

1.1 Ιστορία του Internet

Το διαδίκτυο (Internet) είναι ένα παγκόσμιο σύστημα από συνδεδεμένες συσκευές, ένα δίκτυο από δίκτυα, που τις επιτρέπει να επικοινωνούν και να αλληλοεπιδρούν μεταξύ τους. Δύο ή παραπάνω συσκευές που συνδέονται μεταξύ τους δημιουργούν ένα δίκτυο. Στο κέντρο κάθε δικτύου βρίσκεται ο διακομιστής (server), ένας γρήγορος υπολογιστής που ελέγχει την πληροφορία ανάμεσα στις συσκευές που υπάρχουν σε αυτό. Ένας πάροχος υπηρεσιών διαδικτύου (ISP – Internet Service Provider) επιτρέπει στους χρήστες την πρόσβαση στο διαδίκτυο μέσω των διακομιστών του. Η σύνδεση αυτή γίνεται κατά κύριο λόγο με γραμμές τηλεφώνου αλλά και με άλλα μέσα όπως δορυφόρους και οπτικές ίνες.

Η ανάγκη για την απομακρυσμένη επικοινωνία μεταξύ υπολογιστών φαίνεται να ξεκινάει στα τέλη της δεκαετίας του 1960 στις ΗΠΑ. Το Αμερικανικό Υπουργείο Αμύνης έχει ιδρύσει την υπηρεσία ARPA (Advanced Research Project Agency) και αυτή με τη σειρά της δημιουργεί το ARPANET, ένα δίκτυο που διασύνδεε πανεπιστήμια και εταιρείες με κύριο στόχο τη μελέτη για τη διατήρηση των επικοινωνιών σε περίπτωση πολέμου. Αργότερα δημιουργούνται και άλλα δίκτυα που όμως χρησιμοποιούν διαφορετικά πρωτόκολλα επικοινωνίας, ξεχωριστά για συγκεκριμένους τύπους υπολογιστών και μια καινούργια ανάγκη για ένα πρωτόκολλο που θα ένωνε όλα τα δίκτυα γεννάται. Το 1974 δημιουργείται το TCP (Transmission Control Protocol) και το 1978 το TCP/IP, μία συλλογή πρωτοκόλλων επικοινωνίας που δίνει λύση σε συγκεκριμένα προβλήματα μεταφοράς δεδομένων και στην οποία βασίζεται και το Διαδίκτυο του σήμερα. Το ARPANET πλέον χρησιμοποιεί TCP/IP και εκατοντάδες πανεπιστήμια από όλο τον κόσμο συνδέονται σε αυτό, με αποτέλεσμα να επιβαρύνει τη λειτουργία του. Το 1984 υλοποιείται το πρώτο DNS (Domain Name System) και πλέον οι υπολογιστές του δικτύου αναγνωρίζονται από αλφαβητικά ονόματα που αντιστοιχούν σε διευθύνσεις IP.

Το 1985 δημιουργείται το NSFNET από το NSF (National Science Foundation) και παρέχει υπολογιστές σε διάφορα πανεπιστήμια για τη σύνδεσή τους σε αυτό. Χιλιάδες δίκτυα από πανεπιστήμια και οργανισμούς συνδέονται πάνω σε αυτό και αρχίζει σιγά σιγά να παίρνει τη μορφή του Διαδικτύου όπως το ξέρουμε σήμερα. Το 1990 το ARPANET καταργείται και το 1993 παρουσιάζεται για πρώτη φορά ο Παγκόσμιος Ιστός (World Wide Web – WWW) από το εργαστήριο CERN της Ελβετίας. Είναι ένα νέο σύστημα στο οποίο οι πληροφορίες

αποθηκεύονται σε μορφή πολυμέσων πέραν του απλού κειμένου και παρουσιάζονται στη μορφή ηλεκτρονικών σελίδων. Το γραφικό περιβάλλον πλέον υπάρχει και κάνει τη σύνδεση στο Internet εύκολη για κάθε χρήστη χρησιμοποιώντας απλά ένα πληκτρολόγιο και ένα ποντίκι. Το 1995 το NSFNET διαλύεται και το φορτίο του μεταφέρεται στους πάροχους υπηρεσιών διαδικτύου για τη σύνδεση στο διαδίκτυο.

1.2 Internet και Ιστοσελίδες

Ιστοσελίδα (web page) είναι είδος εγγράφου που μπορούμε να δούμε σε έναν περιηγητή ιστού (browser) και περιέχει πληροφορίες σε μορφή κειμένου, εικόνας, ήχου, βίντεο ή υπερσυνδέσμου. Υπερσύνδεσμος (hyperlink) ονομάζεται το σημείο αυτό από το οποίο μπορούμε να μεταβούμε σε άλλα σημεία της ίδια της σελίδας ή ακόμα και άλλης. Στην ουσία είναι μια μέθοδος παράκαμψης της γραμμικότητας ενός εγγράφου. Ιστότοπος (website) ονομάζεται η συλλογή πολλών ιστοσελίδων κάτω από την ίδια διεύθυνση ή όνομα.

Στις μέρες μας οι ιστοσελίδες δε χρησιμοποιούνται μόνο για διαμοιρασμό πληροφοριών αλλά και για παροχή υπηρεσιών όπως online shopping, video streaming, gaming και άλλα. Η διαδικτυακή εφαρμογή (web application) είναι ένα είδος λογισμικού που τρέχει στον browser και συνήθως είναι υπεύθυνο για τις παραπάνω λειτουργίες. Τα μεγαλύτερα πλεονεκτήματα της διαδικτυακής εφαρμογής απέναντι σε μια κλασσική εφαρμογή είναι ότι δε χρειάζεται εγκατάσταση και είναι λειτουργική στην πλειονότητα των συσκευών που έχουν πρόσβαση στο διαδίκτυο. Για παράδειγμα η ίδια εφαρμογή που εκτελεί μια συγκεκριμένη λειτουργία πρέπει να σχεδιαστεί ξεχωριστά για smartphone, για υπολογιστή που τρέχει Windows και για υπολογιστή που τρέχει Linux ενώ η σχεδίαση της διαδικτυακής εφαρμογής θα γίνει μία φορά και θα τρέχει εξαιρετικά και στις 3 προαναφερθείσες συσκευές.

Οι έννοιες web page, web site και web app πολλές φορές μπερδεύονται και οι άνθρωποι αναφερόμαστε σε αυτές εννοώντας το ίδιο πράγμα. Οπότε συνοψίζοντας κρατάμε ότι web page είναι η μεμονωμένη σελίδα που βλέπουμε στον browser μας, web site είναι το σύνολο των σελίδων κάτω από το ίδιο όνομα που έχουν άμεση σύνδεση μεταξύ τους και web application είναι η εφαρμογή με λειτουργικότητα που παραπέμπει σε εφαρμογή επιτραπέζιου υπολογιστή, τρέχει στον browser και συνήθως φιλοξενείτε σε κάποια ιστοσελίδα.

1.3 Front-end και back-end

Ο Προγραμματισμός Ιστού (Web Development) χωρίζεται σε δύο μεγάλες κατηγορίες. Το Front-end (μπροστινό μέρος) και το Back-end (πίσω μέρος). Για να λειτουργήσει αποτελεσματικά μία ιστοσελίδα, τα δύο αυτά μέρη πρέπει να επικοινωνούν και να συνεργάζονται αποτελεσματικά σαν μία ενιαία μονάδα.

Το Front-end είναι το μέρος της ιστοσελίδας που ο χρήστης αλληλοεπιδρά άμεσα. Πολλές φορές αναφέρεται και ως το μέρος πελάτη (client side) της εφαρμογής. Είναι οτιδήποτε ο χρήστης βλέπει απευθείας στην οθόνη του, δηλαδή κείμενα, χρωματισμοί και styling, φωτογραφίες, πίνακες, μενού πλοήγησης κλπ. Παρέχει τη δομή, την εμφάνιση, τη συμπεριφορά και το περιεχόμενο από οτιδήποτε εμφανίζεται στην οθόνη μας τη στιγμή που ανοίγουμε τον browser. Οι κύριες γλώσσες που απαρτίζουν το front-end είναι οι: HTML (Hypertext Markup Language), CSS (Cascading Style Sheets) και Javascript.

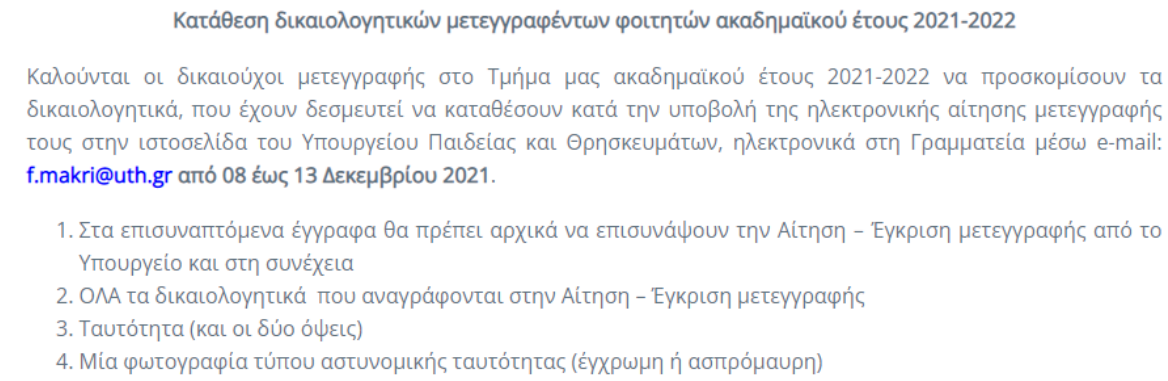
Το Back-end είναι το μέρος της ιστοσελίδας που ο χρήστης δε μπορεί να δει. Το συναντάμε και σαν μέρος εξυπηρετητή (server side). Σε αυτό το μέρος γίνεται η οργάνωση και η αποθήκευση δεδομένων όπως επίσης και η εξασφάλιση ότι οτιδήποτε βρίσκεσαι στο front-end λειτουργεί με σωστό τρόπο. Ο χρήστης παρόλο που δε βλέπει τι συμβαίνει σε αυτό το μέρος, έχει έμμεση πρόσβαση σε κομμάτια του μέσω του front-end. Οι κύριες γλώσσες προγραμματισμού για το back-end είναι οι: PHP, Java, Javascript και Python.

2. HTML, CSS και Javascript

2.1 Εισαγωγή

Όταν μιλήσαμε για το front-end αναφερθήκαμε στην HTML σαν μια από τις κυριότερες γλώσσες προγραμματισμού που το πλαισιώνουν. Η αλήθεια είναι ότι δεν είναι ακριβώς γλώσσα προγραμματισμού, αλλά γλώσσα προσδιορισμού των χαρακτηριστικών που απαρτίζουν μία ιστοσελίδα. Αν παρομοιάζαμε μια ιστοσελίδα με το ανθρώπινο σώμα, τότε η HTML θα ήταν ο σκελετός, το CSS θα ήταν το δέρμα και τα εξωτερικά χαρακτηριστικά και η Javascript θα ήταν το σύνολο των εσωτερικών οργάνων, απαραίτητα για τη ζωή και τη λειτουργικότητα του σώματος.

Ας ρίξουμε μια ματιά σε μία τυχαία ανακοίνωση του Πανεπιστημίου Θεσσαλίας:



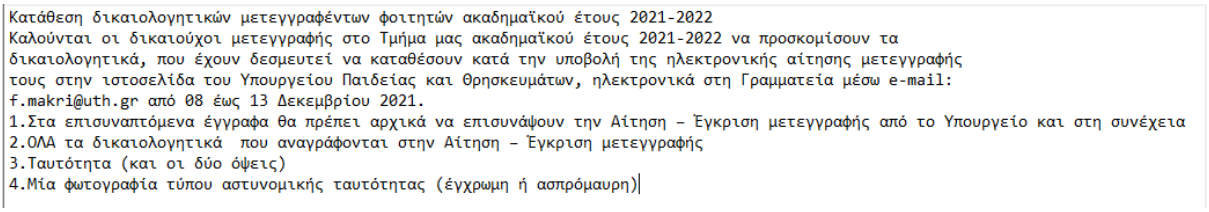
Κατάθεση δικαιολογητικών μετεγγραφέντων φοιτητών ακαδημαϊκού έτους 2021-2022

Καλούνται οι δικαιούχοι μετεγγραφής στο Τμήμα μας ακαδημαϊκού έτους 2021-2022 να προσκομίσουν τα δικαιολογητικά, που έχουν δεσμευτεί να καταθέσουν κατά την υποβολή της ηλεκτρονικής αίτησης μετεγγραφής τους στην ιστοσελίδα του Υπουργείου Παιδείας και Θρησκευμάτων, ηλεκτρονικά στη Γραμματεία μέσω e-mail: f.makri@uth.gr από 08 έως 13 Δεκεμβρίου 2021.

1. Στα επισυναπτόμενα έγγραφα θα πρέπει αρχικά να επισυνάψουν την Αίτηση – Έγκριση μετεγγραφής από το Υπουργείο και στη συνέχεια
2. ΟΛΑ τα δικαιολογητικά που αναγράφονται στην Αίτηση – Έγκριση μετεγγραφής
3. Ταυτότητα (και οι δύο όψεις)
4. Μία φωτογραφία τύπου αστυνομικής ταυτότητας (έγχρωμη ή ασπρόμαυρη)

Εικόνα 1 - Ανακοίνωση Πανεπιστημίου Θεσσαλίας

Όλες οι ανακοινώσεις, όπως και τα περισσότερα κομμάτια κειμένου που βρίσκονται σε ηλεκτρονική ή φυσική μορφή, χρησιμοποιούν ένα συγκεκριμένο μοτίβο μορφοποίησης. Η συγκεκριμένη περιέχει τίτλο, ξεχωριστή παράγραφο με κενό πριν και μετά από αυτή, έντονα γράμματα και μία αριθμημένη λίστα. Αν αφαιρούσαμε όλα αυτά, αυτό που θα έμενε θα ήταν απλά κείμενο και η ανακοίνωση θα έμοιαζε κάπως έτσι:



```
Κατάθεση δικαιολογητικών μετεγγραφέντων φοιτητών ακαδημαϊκού έτους 2021-2022
Καλούνται οι δικαιούχοι μετεγγραφής στο Τμήμα μας ακαδημαϊκού έτους 2021-2022 να προσκομίσουν τα
δικαιολογητικά, που έχουν δεσμευτεί να καταθέσουν κατά την υποβολή της ηλεκτρονικής αίτησης μετεγγραφής
τους στην ιστοσελίδα του Υπουργείου Παιδείας και Θρησκευμάτων, ηλεκτρονικά στη Γραμματεία μέσω e-mail:
f.makri@uth.gr από 08 έως 13 Δεκεμβρίου 2021.
1.Στα επισυναπτόμενα έγγραφα θα πρέπει αρχικά να επισυνάψουν την Αίτηση – Έγκριση μετεγγραφής από το Υπουργείο και στη συνέχεια
2.ΟΛΑ τα δικαιολογητικά  που αναγράφονται στην Αίτηση – Έγκριση μετεγγραφής
3.Ταυτότητα (και οι δύο όψεις)
4.Μία φωτογραφία τύπου αστυνομικής ταυτότητας (έγχρωμη ή ασπρόμαυρη)
```

Εικόνα 2 - Ανακοίνωση Πανεπιστημίου Θεσσαλίας χωρίς μορφοποίηση

Αυτό ακριβώς κάνουμε και με τη χρήση της HTML. Παίρνουμε ένα κείμενο και χρησιμοποιώντας ειδική σήμανση και κανόνες, δημιουργούμε δομή ώστε να γίνεται κατανοητό από τον αναγνώστη. Η δομή αυτή επιτυγχάνεται χρησιμοποιώντας ετικέτες (tags). Τα tags είναι λέξεις-κλειδιά, που αρχίζουν και τελειώνουν με το σύμβολο του μεγαλύτερου (>) και του μικρότερου (<) και προσδιορίζουν πως ο browser απεικονίζει το περιεχόμενο της σελίδας. Ένα ανοιχτό και ένα κλειστό μέρος είναι απαραίτητα για τα περισσότερα tags πχ   που προσδιορίζουν μια παράγραφο ενώ υπάρχουν και tags χωρίς κλειστό μέρος πχ <img> που προσδιορίζει μία εικόνα. Το κάθε κομμάτι που περικλείεται από tags ονομάζεται στοιχείο (element) και κάθε στοιχείο μπορεί να έχει μία ή παραπάνω ιδιότητες (attributes) για να αποδώσει επιπρόσθετες πληροφορίες που σχετίζονται με αυτό. Παρακάτω φαίνεται η βασική και απαραίτητη δομή μιας HTML σελίδας:

```
<!DOCTYPE html>
<html>
<head>
  <title>Example Page</title>
</head>
<body>
  <img src=”image_example.jpg” width=”200”>
</body>
</html>
```

Το <!DOCTYPE html> προσδιορίζει τη χρήση της τελευταίας έκδοσης HTML στον κώδικά μας. Το <html> </html> είναι το ριζικό στοιχείο. Όλα τα υπόλοιπα στοιχεία περικλείονται από αυτό. Το <head> </head> περιέχει πληροφορίες για τη σελίδα. Το <title> </title> περιέχει τον τίτλο της σελίδας, ο οποίος εμφανίζεται στην καρτέλα του browser. Στο <body> </body> βρίσκονται όλα τα περιεχόμενα της σελίδας, στην συγκεκριμένη περίπτωση μία εικόνα <img>. Τα src και width είναι attributes και προσδιορίζουν το όνομα και το πλάτος της αντίστοιχα.

2.2 CSS

Το CSS το χρησιμοποιούμε για να δώσουμε στυλ και εμφάνιση σε μια ιστοσελίδα. Η δήλωση των στυλ γίνεται είτε απ’ ευθείας στο σημείο που ανοίγει το HTML tag, είτε στο head μέρος της σελίδας ή σε ένα εξωτερικό αρχείο που συνδέεται άμεσα με τη σελίδα μας. Το πλεονέκτημα της τελευταίας μεθόδου είναι πως μπορούμε να δώσουμε κοινά στυλ σε πολλές σελίδες χωρίς να αντιγράφουμε τον ίδιο κώδικα ξανά και ξανά, απλά συνδέοντας το ίδιο αρχείο σε κάθε σελίδα. Το συντακτικό του είναι πολύ απλό και βασίζεται σε κανόνες που εφαρμόζουμε ομαδοποιώντας στοιχεία της σελίδας. Για παράδειγμα, ο παρακάτω κώδικας δηλώνει μπλε χρώμα και 50 pixels μέγεθος γραμματοσειράς για όλες τις επικεφαλίδες μορφής h1.

```
h1{
  color: blue;
  font-size: 50px;
}
```

Αναλύοντας το συγκεκριμένο παράδειγμα, το h1 έχει το ρόλο του επιλογέα (selector), το color και το font-size είναι οι ιδιότητες (properties) και τα blue και 50px είναι οι τιμές (values).

Ομαδοποίηση, εκτός από το όνομα του στοιχείου απ' ευθείας, γίνεται και με το id selector, όπου συμβολίζεται με την δίεση (#) ακολουθούμενο από ένα όνομα καθώς και με το class selector όπου συμβολίζεται με την τελεία (.) ακολουθούμενη από ένα όνομα. Η δήλωσή τους γίνεται στο πρώτο μέρος των tags με τη μορφή attribute. Κάθε στοιχείο μπορεί να έχει το πολύ ένα id ενώ δεν υπάρχει περιορισμός για τα classes. Επίσης αν δύο ή παραπάνω στυλ συγκρούονται στη δήλωσή τους, αυτό που θα εφαρμοστεί είναι πρώτα το id αφού έχει τη μεγαλύτερη βαρύτητα, μετά το class και τέλος το στοιχείο. Αν έχουν την ίδια βαρύτητα τότε θα εφαρμοστεί αυτό που δηλώθηκε τελευταίο.

2.3 Javascript

Η Javascript είναι μία γλώσσα σεναρίων (script language) και μαζί με τις γλώσσες που αναφέραμε παραπάνω συμπληρώνει τα βασικά συστατικά για τη δημιουργία μιας μοντέρνας ιστοσελίδας. Μας επιτρέπει να προσθέσουμε λειτουργικότητα στην ιστοσελίδα μας και να εμφανίσουμε δυναμικό περιεχόμενο. Χαρακτηριστικά όπως ένα μενού που αναδύεται όταν πατάμε ένα κουμπί ή μια κυκλική παρουσίαση φωτογραφιών επιτυγχάνονται με τη χρήση της Javascript. Η πιο κοινή μορφή της γλώσσας που συναντάμε είναι αυτή του client και τρέχει μέσω του browser ωστόσο όπως θα δούμε παρακάτω μπορούμε να τρέξουμε κώδικα Javascript και σε server.

Υπάρχουν 3 τρόποι για να προσθέσουμε Javascript λειτουργικότητα στα HTML αρχεία μας. Αρχικά μπορούμε να το κάνουμε στην ίδια γραμμή του στοιχείου. Ο παρακάτω κώδικας δημιουργεί ένα κουμπί, το οποίο αν το πατήσουμε θα εμφανιστεί στην οθόνη μας το μήνυμα «Example button clicked!».

```
<button onclick="alert ('Example button clicked!')">Example
```

Μπορούμε με τη βοήθεια του script tag να ενσωματώσουμε κώδικα μέσα στο HTML αρχείο. Ο κώδικας αυτός εκτελείτε τη στιγμή που φορτώσουμε την σελίδα και εμφανίζει το μήνυμα «Hello world!»

```
<script>
   function () {
      alert ("Hello world!")
   }
</script>
```

Τέλος, μπορούμε να αναπτύξουμε τον κώδικα σε διαφορετικό αρχείο και να τον συμπεριλάβουμε στη σελίδα μας πάλι με τη χρήση του script tag:

```
<script src="javascript.js"></script>
```

Η Javascript, όπως οι περισσότερες γλώσσες προγραμματισμού, υποστηρίζει συγκεκριμένους τύπους δεδομένων, μεταβλητές, σχόλια, συναρτήσεις, δομές επανάληψης, δομές ελέγχου αληθείας κλπ. Οι δυνατότητές της είναι τεράστιες και αυτή τη στιγμή είναι μία από τις πιο δημοφιλείς γλώσσες ανάπτυξης λογισμικού παγκοσμίως.

3. Σχεδιασμός εφαρμογής

3.1 Ανάλυση λειτουργιών

Η ιδέα ήταν να δημιουργήσουμε ένα website το οποίο λειτουργεί σαν ευρετήριο εστιατορίων που παρέχει μια λίστα με τα αποθηκευμένα εστιατόρια για προβολή καθώς και την επιλογή της αναζήτησης. Επίσης ο χρήστης μπορεί να γίνει μέλος και να προσθέσει το δικό του αγαπημένο εστιατόριο ή να βαθμολογήσει και να αφήσει την κριτική του σε άλλα. Άρα, συνοψίζοντας τα παραπάνω και προχωρώντας στην ανάπτυξη της εφαρμογής μας, οι βασικές λειτουργίες που θέλουμε να συναντήσουμε είναι οι εξής:

1. Εμφάνιση εστιατορίων
2. Αναζήτηση εστιατορίων
3. Προβολή ενός εστιατορίου για περισσότερες πληροφορίες
4. Εγγραφή χρήστη
5. Σύνδεση χρήστη
6. Προσθήκη/επεξεργασία/διαγραφή εστιατορίου (μόνο εγγεγραμμένοι χρήστες)
7. Προσθήκη/διαγραφή κριτικής και βαθμολογίας σε κάθε εστιατόριο ξεχωριστά (μόνο εγγεγραμμένοι χρήστες)

Πιο λεπτομερώς, ανοίγοντας το website, βρίσκουμε την αρχική σελίδα (home) όπου περιέχει στην κορυφή μια μπάρα πλοήγησης για τη μετάβαση στις υπόλοιπες σελίδες, ένα μήνυμα καλωσορίσματος και ένα πεδίο αναζήτησης όπου ο χρήστης μπορεί να πληκτρολογήσει για αναζήτηση εστιατορίου με οποιοδήποτε κριτήριο (πχ. όνομα, πόλη). Η

μπάρα πλοήγησης θα έχει τις επιλογές “Home”, “Restaurants”, “Add Restaurant”, “Login”, “Register”.

Πηγαίνοντας στη σελίδα “Restaurants” βρίσκουμε έναν χάρτη με όλα τα αποθηκευμένα εστιατόρια σημαδεμένα πάνω σε αυτόν με κουκίδες. Ο χάρτης είναι παγκόσμιος και ομαδοποιεί τα εστιατόρια που βρίσκονται σε κάθε περιοχή ανάλογα με το πλήθος τους. Κάνοντας μεγέθυνση του χάρτη εμφανίζεται η ακριβής τοποθεσία τους και πατώντας πάνω τους εμφανίζεται η επωνυμία και η διεύθυνση τους. Κάτω από τον χάρτη έχουμε μια λίστα με τα αποθηκευμένα εστιατόρια όπου εμφανίζεται μια μικρή φωτογραφία, το όνομα, μια σύντομη περιγραφή και η διεύθυνση του εκάστοτε εστιατορίου. Επίσης υπάρχει ένα κουμπί για προβολή περισσότερων πληροφοριών.

Πατώντας το κουμπί περισσότερων πληροφοριών, πηγαίνουμε στη σελίδα προβολής του εστιατορίου. Εκεί συναντάμε τι φωτογραφίες του εστιατορίου, το όνομα, τη βαθμολογία, την περιγραφή, τη διεύθυνση, τις κουζίνες που παρέχει, το κόστος για δείπνο ενός ατόμου σε ένα εύρος τιμών, την ιστοσελίδα, το τηλέφωνο και το email του. Πιο κάτω υπάρχουν οι κριτικές που έχουν προστεθεί για το συγκεκριμένο εστιατόριο, το όνομα του χρήστη που τις πρόσθεσε, η βαθμολογία που άφησε και η ημερομηνία που προστέθηκαν. Επίσης σε αυτή τη σελίδα εμφανίζεται ένας μικρός χάρτης με την τοποθεσία του εστιατορίου καρφιτσωμένη πάνω του.

Πατώντας το κουμπί προσθήκης εστιατορίου διαπιστώσουμε ότι εμφανίζεται στην οθόνη μας ένα μήνυμα σφάλματος το οποίο αναφέρει ότι πρέπει να έχουμε κάνει πρώτα σύνδεση και μας μεταβιβάζει στην κατάλληλη σελίδα. Σε περίπτωση που δεν έχουμε λογαριασμό πρέπει πρώτα να κάνουμε εγγραφή. Κάνοντας σύνδεση λοιπόν, αποκτάμε πρόσβαση στη σελίδα “Add Restaurant” και συναντάμε μια φόρμα συμπλήρωσης με τα εξής πεδία:

- Τίτλος
- Διεύθυνση
- Ταχυδρομικός Κώδικας
- Πόλη
- Χώρα
- Ανέβασμα φωτογραφιών
- Κατώτατη τιμή γεύματος
- Ανώτατη τιμή γεύματος
- Ιστοσελίδα
- Τηλέφωνο
- Email
- Επιλογή από λίστα κουζινών που παρέχονται

- Περιγραφή

Αφού συμπληρώσουμε τη φόρμα με όλα τα απαραίτητα πεδία, μεταβαίνουμε στη σελίδα προβολής του νέου μας εστιατορίου. Επίσης παρατηρούμε ότι εφόσον προσθέσαμε ένα δικό μας εστιατόριο, έχουμε την επιλογή να το ενημερώσουμε για να αλλάξουμε τυχόν λανθασμένη πληροφορία ή να το διαγράψουμε. Αυτές οι 2 επιλογές εμφανίζονται μόνο στην περίπτωση που είμαστε ιδιοκτήτες της δημοσίευσης και μόνο για τα δικά μας εστιατόρια.

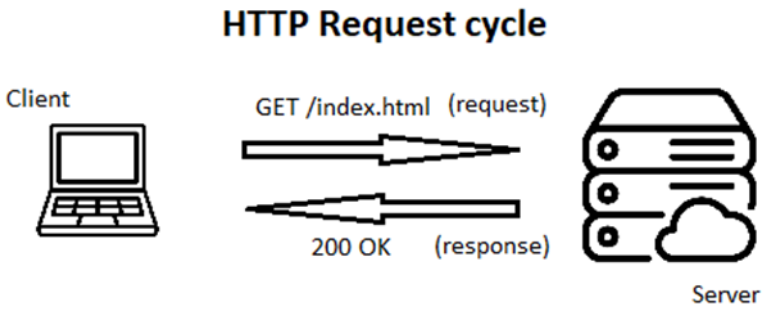
Έχοντας κάνει σύνδεση παρατηρούμε μία ακόμα λειτουργία που δεν έχουν οι επισκέπτες, αυτή της προσθήκης κριτικής. Πηγαίνοντας στη σελίδα προβολής εστιατορίου, εμφανίζεται το πεδίο βαθμολόγησης και προσθήκης σχολίων όπου μπορούμε να συμπληρώσουμε και να προσθέσουμε. Επίσης υπάρχει επιλογή για να σβήσουμε τη δική μας κριτική σε περίπτωση που αλλάξουμε γνώμη αλλά δεν έχουμε καμία επιλογή για να πειράξουμε κριτικές άλλων χρηστών.

3.2 Node.js και Express

Όταν μιλήσαμε για Javascript, αναφέραμε ότι είναι η γλώσσα προγραμματισμού που δίνει λογική στη σελίδα μας. Τρέχει στον browser που χρησιμοποιούμε και ασχολείται με το client κομμάτι της εφαρμογής. Αλλά δεν είναι μόνο αυτό. Το Node.js [1] είναι ένα περιβάλλον λειτουργίας ανοιχτού κώδικα (open source runtime environment) που μας επιτρέπει να εκτελούμε Javascript κώδικα εκτός του browser. Αυτό το καθιστά κατάλληλο για την server-side ανάπτυξη διαδικτυακών εφαρμογών. Είναι εξαιρετικά γρήγορο στην εκτέλεση του κώδικα. Έχει τεράστια βιβλιοθήκη πακέτων (πάνω από 1.000.000) που υποστηρίζεται από μεγάλη κοινότητα για την προσθήκη λειτουργιών στην εφαρμογή μας μέσω του npm (Node Package Manager). Επίσης διαθέτει ενσωματωμένη λειτουργικότητα για τη μεταφορά δεδομένων μέσω HTTP (HyperText Transfer Protocol). Για να χρησιμοποιήσουμε το Node.js πρέπει πρώτα να το εγκαταστήσουμε και έπειτα να το τρέξουμε από το τερματικό.

Το HTTP είναι το βασικό πρωτόκολλο επικοινωνίας που χρησιμοποιούν οι browser για τη μεταφορά δεδομένων ανάμεσα σε server και client. Αυτό επιτυγχάνεται μέσω ενός μοντέλου αιτήσεων-απαντήσεων (request-response). Οι πιο βασικές μέθοδοι αιτήσεων είναι η GET και η POST. Τη μέθοδο GET τη χρησιμοποιούμε όταν θέλουμε να κάνουμε μια αίτηση αποστολής ενός αρχείου από τον server προς τον client (πχ ένα HTML αρχείο) ενώ τη μέθοδο POST τη χρησιμοποιούμε για την αποστολή δεδομένων στο server (πχ username και password για

login). Και στις δύο περιπτώσεις, λαμβάνουμε πίσω ένα response το οποίο εμπεριέχει έναν κωδικό που μας πληροφορεί για την κατάστασή του (θετικό, αρνητικό κλπ).



Εικόνα 3 - HTTP Request cycle

Το Express [2] είναι ένα πακέτο του Node.js, ένα framework που μας βοηθάει να φτιάχνουμε διαδικτυακές εφαρμογές. Framework ονομάζεται το λογισμικό το οποίο έχει δημιουργηθεί για να υποστηρίξει την ανάπτυξη μιας εφαρμογής, διαχειρίζοντας λειτουργίες χαμηλού επιπέδου, γλιτώνοντάς μας έτσι από επιπλέον χρόνο και πιθανά σφάλματα. Τα βασικά χαρακτηριστικά του Express, αυτά για το οποίο θα το χρησιμοποιήσουμε εμείς, είναι για να:

- Χειριζόμαστε requests
- Δημιουργούμε server έτοιμο να ακούσει requests
- Ετοιμάζουμε response
- Αντιστοιχίσουμε το κάθε request/response στη δική του διαδρομή (route)

3.3 EJS

Το EJS (Embedded JavaScript) [3] είναι μια γλώσσα προτύπου (templating language) που χρησιμοποιείται στη Node όταν δουλεύουμε με Express και επιτρέπει την παραγωγή HTML κώδικα χρησιμοποιώντας Javascript. Για να γίνει κατανοητή η έννοια του προτύπου, ας πούμε ότι θέλουμε να εμφανίσουμε στη σελίδα μας μία λίστα δεδομένων από τη βάση. Τα δεδομένα πρέπει να παρουσιάζονται με ένα συγκεκριμένο στυλ, το οποίο θα το ορίσουμε μία φορά. Έπειτα θα περικλείσουμε τον HTML κώδικα σε μια δομή επανάληψης Javascript με τα ειδικά EJS tags και το περιεχόμενο θα παραχθεί δυναμικά ανάλογα με τα δεδομένα που ανακτώνται κάθε φορά από τη βάση και τα κριτήρια της δομής επανάληψης.

```
<% for(let name of names){ %>
  <p> <%= name %> </p>
<% } %>
```

Για το παραπάνω παράδειγμα, έστω ότι έχουμε ανακτήσει τον πίνακα names από τη βάση. Ανοίγουμε μια δομή επανάληψης όπου θα προσπελάσει όλα τα δεδομένα του πίνακα και έπειτα θα τα εμφανίσει σε μορφή παραγράφων. Το παράδειγμα μας κάνει χρήση και των δύο πιο συχνών tag. Το "<%" tag χρησιμοποιείται για έλεγχο καταστάσεων και ροής. Το "<%=" tag χρησιμοποιείται για έξοδο δεδομένων και στο συγκεκριμένο παράδειγμα κάνει εμφάνιση της μεταβλητής "name". Τέλος να αναφέρουμε ότι για να γίνει έγκυρη χρήση του EJS στα HTML αρχεία, η κατάληξη αυτών θα πρέπει από .html να μετονομαστεί σε .ejs.

3.4 Βάση δεδομένων: MongoDB

Στον προγραμματισμό, οι βάσεις δεδομένων (databases) χωρίζονται σε δύο μεγάλες κατηγορίες, τις SQL (Structured Query Language) databases και τις NoSQL databases. Στην πρώτη κατηγορία αναφερόμαστε σε σχεσιακές βάσεις δεδομένων (πχ. Oracle, MySQL) όπου πρέπει να προκαθορίσουμε ένα σχήμα από πίνακες (tables) πριν εισχωρήσουμε οτιδήποτε, ώστε όλα τα δεδομένα να είναι σε αυτή τη μορφή. Οι βάσεις της δεύτερης κατηγορίας δεν έχουν αυτή την προϋπόθεση και τα δεδομένα σε αυτές μπορούν να αποθηκεύονται με άλλους τρόπους. Ο πιο δημοφιλής τρόπος είναι αυτός του εγγράφου (document).

Η MongoDB [4] είναι μια document-oriented database όπου η αποθήκευση των δεδομένων γίνεται σε έγγραφα που μοιάζουν με έγγραφα JSON. JSON είναι ένα είδος κειμένου που χρησιμοποιείται για τη μετάδοση αντικειμένων (objects) δεδομένων. Ο λόγος που χρησιμοποιήσαμε τη Mongo στη συγκεκριμένη εργασία είναι επειδή είναι πολύ γρήγορη και γιατί δουλεύει εξαιρετικά καλά σε εφαρμογές που είναι στημένες με Node και Express. Για να την αξιοποιήσουμε, πρέπει πρώτα να την εγκαταστήσουμε και έπειτα να την τρέξουμε μέσα από το τερματικό.

Για να συνδέσουμε τη Mongo με την εφαρμογή μας, δηλαδή με την Express, χρησιμοποιήσαμε ένα εργαλείο που ονομάζεται Mongoose [5]. Η Mongoose είναι μια ODM (Object Data Modeling) βιβλιοθήκη που μας επιτρέπει να δημιουργήσουμε ένα σχήμα (Schema) με καθορισμένους τύπους δεδομένων και έπειτα να κατασκευάσουμε ένα μοντέλο (Model) βασισμένο στο συγκεκριμένο σχήμα. Το μοντέλο αυτό στη συνέχεια μπορεί πολύ εύκολα να μεταφραστεί σε ένα MongoDB έγγραφο. Επίσης, εφόσον έχουμε προκαθορίσει τα

σχήματα και μοντέλα μας, η Mongoose διαθέτει συναρτήσεις για να κάνουμε επαλήθευση δεδομένων και να δημιουργήσουμε δικές μας απλές ή πολύπλοκες φράσεις ανάκτησης αυτών (data queries). Με απλά λόγια λειτουργεί σαν διαμεσολαβητής ανάμεσα στη βάση δεδομένων και το server μέρος της εφαρμογής.

3.5 Bootstrap

Το Bootstrap [6] είναι το πιο δημοφιλές HTML, CSS και Javascript framework ανοιχτού κώδικα για την ανάπτυξη ευπαρουσίαστων και άμεσα αποκρινόμενων σε κάθε μέγεθος οθόνης (responsive) ιστοσελίδων. Στην ουσία είναι μια μεγάλη συλλογή από επαναχρησιμοποιούμενα και παραμετροποιήσιμα κομμάτια κώδικα, γραμμένα σε CSS και Javascript τα οποία μπορούμε πολύ εύκολα να εντάξουμε σε έναν HTML κώδικα. Με τη βοήθεια προκαθορισμένων κλάσεων μπορούμε να δώσουμε στυλ ή να δημιουργήσουμε οντότητες όπως ένα κουμπί ή μια μπάρα πλοήγησης πολύ γρήγορα και εύκολα, κερδίζοντας έτσι πολύτιμο χρόνο για την ανάπτυξη των λειτουργιών της εφαρμογής.

4. Ανάπτυξη της εφαρμογής

4.1 Βασική λειτουργικότητα

Έχοντας εγκαταστήσει το Node.js στον υπολογιστή μας, το πρώτο πράγμα που κάναμε ήταν να εγκαταστήσουμε τα απαραίτητα πακέτα με το npm από το terminal, δηλαδή express [7], mongoose [8] και ejs [9]. Δημιουργήσαμε το αρχείο app.js όπου θα ήταν και η αφετηρία του προγράμματός μας. Κάναμε τις κατάλληλες δηλώσεις για να μπορούμε να χρησιμοποιήσουμε τα πακέτα που εγκαταστήσαμε. Χρησιμοποιήσαμε τη συνάρτηση app.listen για να δεσμεύσουμε ένα συγκεκριμένο port για την εφαρμογή μας ώστε να ξεκινήσει να ακούει για αιτήσεις και τη συνάρτηση app.get με παράμετρο “/” για να διαχειριστούμε ένα get request το οποίο θα απαντάει με το home page της εφαρμογής μας. Επίσης δημιουργήσαμε και το αρχείο home.ejs το οποίο για την ώρα θα μείνει κενό. Ο server έχει στηθεί.

```
const express = require("express");
const app = express();

app.get('/', (req,res) => {
   res.render('home');
})

app.listen(3000, ()=> {
   console.log('We are on port 3000');
})
```

Έπειτα δημιουργήσαμε ένα ξεχωριστό αρχείο restaurant.js που περιέχει το σχήμα του εκάστοτε εστιατορίου, δηλαδή τις ιδιότητες τις οποίες έχει και παράλληλα ενημερώσαμε το app.js για να γίνει σύνδεση μεταξύ mongoose και mongo. Το νέο αρχείο αρχικά μοιάζει κάπως έτσι:

```
const mongoose = require('mongoose');

const RestaurantSchema = new Schema({
   title: String,
   description: String
})

module.exports = mongoose.model('Restaurant', RestaurantSchema);
```

Στη συνέχεια θέλαμε να εισάγουμε RESTful routes στην εφαρμογή μας. Το REST (Representational State Transfer) είναι ένα σύνολο κανόνων που χρησιμοποιείται από διάφορες γλώσσες προγραμματισμού και στοχεύει στο να αντιστοιχίσει HTTP μεθόδους και CRUD (Create, Read, Update, Delete) λειτουργικότητα σε ένα μοτίβο το οποίο είναι εύχρηστο και κατανοητό τόσο από προγραμματιστές όσο και από χρήστες. Όπως παρατηρούμε και στην περίπτωση των εστιατορίων, θέλουμε να υπάρχει λειτουργικότητα για δημιουργία, προβολή, ενημέρωση και διαγραφή.

Δημιουργήσαμε έναν καινούργιο φάκελο “views” ο οποίος περιέχει τις διαφορετικές σελίδες για διαφορετική λειτουργία όπου μπορεί να επισκεφθεί ο χρήστης. Οι σελίδες είναι οι εξής:

- index.ejs: Λίστα με όλα τα εστιατόρια
- show.ejs: Περισσότερες πληροφορίες για ένα εστιατόριο
- new.ejs: Φόρμα δημιουργίας νέου εστιατορίου
- edit.ejs: Φόρμα ενημέρωσης για ήδη υπάρχον εστιατόριο

Τα αντίστοιχα routes που προσθέσαμε στο app.js είναι τα εξής:

```
app.get('/restaurants', async (req, res) => {
  const restaurants = await Restaurant.find({});
  res.render('restaurants/index', { restaurants })
});

app.get('/restaurants/new', (req, res) => {
  res.render('restaurants/new');
})

app.post('/restaurants', async (req, res) => {
  const restaurant = new Restaurant(req.body.restaurant);
  await restaurant.save();
  res.redirect(`/restaurants/${restaurant._id}`)
})

app.get('/restaurants/:id', async (req, res,) => {
  const restaurant = await Restaurant.findById(req.params.id)
  res.render('restaurants/show', { restaurant });
});

app.get('/restaurants/:id/edit', async (req, res) => {
  const restaurant = await Restaurant.findById(req.params.id)
  res.render('restaurants/edit', { restaurant });
})

app.put('/restaurants/:id', async (req, res) => {
  const { id } = req.params;
  const restaurant = await Restaurant.findByIdAndUpdate(id, {
...req.body.restaurant });
  res.redirect(`/restaurants/${restaurant._id}`)
});

app.delete('/restaurants/:id', async (req, res) => {
  const { id } = req.params;
  await Restaurant.findByIdAndDelete(id);
  res.redirect('/restaurants');
})
```

Για το πρώτο route, η συνάρτηση Restaurant.find({}) δεν περιέχει κάποιο όρισμα άρα επιστρέφει όλα τα αποθηκευμένα εστιατόρια και τα αποθηκεύει στη μεταβλητή restaurants. Η μεταβλητή αυτή περνάει ως όρισμα στη συνάρτηση res.render για το index page, άρα μπορούμε να τη διαχειριστούμε όταν βρισκόμαστε στο index.ejs αρχείο. Όπως είπαμε και προηγουμένως, μια απλή δομή επανάληψης αρκεί για να εμφανίσει όλες τις καταχωρήσεις.

Το δεύτερο route μας εμφανίζει τη σελίδα με τη φόρμα δημιουργίας νέου εστιατορίου, new.ejs. Η φόρμα υποβάλλει δεδομένα με τη μέθοδο POST διότι δε θέλουμε να προβάλουμε κάτι αλλά να στείλουμε κάτι στον server. Επίσης σε κάθε στοιχείο της φόρμας προσθέσαμε τη λέξη-κλειδί required ώστε να μη μπορεί ο χρήστης να την υποβάλλει αν δεν έχει συμπληρώσει σωστά τα απαραίτητα πεδία.

Το τρίτο route χρησιμοποιεί την εντολή new Restaurant με όρισμα το req.body.restaurant για να αντλήσει τα δεδομένα που υποβάλλαμε μέσω της φόρμας της new.ejs σελίδας και να τα αποθηκεύσει στη βάση. Έπειτα μας ανακατευθύνει στη σελίδα show.ejs. Η μεταβλητή restaurant._id που χρησιμοποιείται στην ανακατεύθυνση είναι μια μοναδική συμβολοσειρά που δημιουργείται αυτόματα από τη Mongo και αντιπροσωπεύει το κάθε αντικείμενο που αποθηκεύεται στη βάση.

Το τέταρτο route αντλεί από τη διεύθυνση (url) της σελίδας το id του εστιατορίου που θέλουμε να προβάλλουμε και το περνάει σαν όρισμα στην συνάρτηση Restaurant.findById. Έχοντας πλέον το εστιατόριο που θέλουμε στη μεταβλητή restaurant, το περνάμε στη σελίδα show.ejs και το εμφανίζουμε.

Το πέμπτο route δέχεται με τον ίδιο τρόπο το id του εστιατορίου από τη διεύθυνση της σελίδας και μας εμφανίζει τη σελίδα με τη φόρμα για ενημέρωση του συγκεκριμένου εστιατορίου. Η φόρμα ενημέρωσης είναι όμοια με τη φόρμα προσθήκης αλλά περιέχει τις ήδη υπάρχουσες τιμές.

Το έκτο route βρίσκει το id του εστιατορίου από τη διεύθυνση και ενημερώνει τα στοιχεία που άλλαξαν. Έπειτα μας ανακατευθύνει στη σελίδα εμφάνισης όπου μπορούμε να δούμε τα ενημερωμένα στοιχεία.

Το έβδομο route βρίσκει το id του εστιατορίου από τη διεύθυνση και το διαγράφει. Η ανακατεύθυνση μετά γίνεται στη σελίδα προβολής όλων των εστιατορίων.

Παρατηρούμε ότι στα routes τα οποία έχουμε να κάνουμε με διαχείριση δεδομένων από και προς τον server, κάνουμε χρήση της λέξης κλειδί async στη δήλωση της συνάρτησης επιστροφής. Όπως είναι γνωστό, η Javascript είναι μία μονο-νηματική (single-threaded) γλώσσα άρα οι γραμμές κώδικα εκτελούνται η μία μετά την άλλη. Μόνο μία λειτουργία εκτελείτε τη φορά και οποιαδήποτε άλλη μπλοκάρεται μέχρι να ολοκληρωθεί η προηγούμενη (synchronous). Αυτό μπορεί να δημιουργήσει πρόβλημα στην περίπτωση που έχουμε να κάνουμε με ανάκτηση δεδομένων από εξωτερική πηγή (server) διότι δεν γνωρίζουμε εξαρχής τον χρόνο που απαιτείται για να ολοκληρωθεί η λειτουργία ανάκτησης. Γι' αυτό πρέπει να χρησιμοποιούμε τη λέξη async ώστε να κάνουμε τον κώδικά μας ασύγχρονο (asynchronous) στη συνάρτηση που θέλουμε να εκτελέσουμε τέτοιες λειτουργίες και τη λέξη await στη

λειτουργία που απαιτεί χρόνο ώστε να παύσει η εκτέλεση του κώδικα και να συνεχίσει μόνο μετά την ολοκλήρωσή της.

Το επόμενο που κάναμε για να τελειώσουμε με τις βασικές λειτουργίες διαχείρισης εστιατορίων, ήταν να προσθέσουμε επικύρωση δεδομένων στην πλευρά του server. Μέχρι στιγμής, η μόνη επικύρωση γινόταν στον browser (client) με τη χρήση της λέξης required στις φόρμες συμπλήρωσης. Ωστόσο υπάρχουν και άλλοι τρόποι υποβολής όπως με κάποιο εργαλείο δοκιμής HTTP requests. Για να είμαστε σίγουροι ότι οι φόρμες μας υποβάλλονται σωστά κάθε φορά, χρησιμοποιήσαμε ένα εργαλείο της Javascript που μας βοηθάει να επικυρώνουμε τα δεδομένα που υποβάλλονται πριν αυτά αποθηκευτούν στο server. Το εργαλείο ονομάζεται Joi [10], το εγκαταστήσαμε στο project μας μέσω του npm και ο τρόπος με τον οποίο δουλεύει είναι προσδιορίζοντας ένα σχήμα με τα δεδομένα που υποβάλλονται και θέτοντας προϋποθέσεις που πρέπει να τηρούνται για το καθένα ώστε να είναι κατάλληλα για αποθήκευση. Στην περίπτωσή μας αρχικά, στη δήλωση του σχήματος, απαιτούμε να βρίσκεται το restaurant object, να υπάρχει τίτλος ο οποίος να είναι της μορφής String, να υπάρχει περιγραφή της μορφής String και να υπάρχει η κατώτατη τιμή δείπνου η οποία είναι της μορφής Number και είναι θετικός αριθμός.

```
restaurantSchema = Joi.object({
    restaurant: Joi.object({
        title: Joi.string().required(),
        description: Joi.string().required(),
        minprice: Joi.number().required().min(0)
    }).required()
});
```

Ο έλεγχος γίνεται με τη μορφή middleware συνάρτησης στα route τα οποία υποβάλλουν οι φόρμες. Αν υπάρχει κάποιο πρόβλημα εμφανίζεται κατάλληλο μήνυμα σφάλματος, διαφορετικά συνεχίζει η διαδικασία αποθήκευσης στον server.

```
validateRestaurant = (req, res, next) => {
    const { error } = restaurantSchema.validate(req.body);
    if (error) {
        const msg = error.details.map(el => el.message).join(',');
        throw new ExpressError(msg, 400);
    } else {
        next();
    }
}
```

4.2 Στυλ εφαρμογής

Προχωρώντας στην ανάπτυξη της εφαρμογής, παρατηρήσαμε ότι δημιουργούνται πολλά αρχεία και πολλές φορές χρειάζεται να επαναλάβουμε τον ίδιο κώδικα. Για να το επιλύσουμε αυτό χρησιμοποιήσαμε ένα εργαλείο που ονομάζεται ejs-mate [11] και προσθέτει επιπλέον λειτουργικότητα στο ejs. Αρχικά το εγκαταστήσαμε μέσω του npm και το συμπεριλάβαμε στο app.js αρχείο μας με την εντολή app.engine('ejs', ejsMate). Στη συνέχεια δημιουργήσαμε ένα αρχείο που ονομάζεται boilerplate.ejs και περιέχει τη βασική δομή ενός html αρχείου με τη μόνη διαφορά ότι ανάμεσα στα body tags θα υπάρχει η εξής γραμμή: <%- body %>. Μετά πήγαμε σε κάθε αρχείο που έχουμε δημιουργήσει ως τώρα και διαγράψαμε τα πάντα εκτός του περιεχομένου εντός των body tags. Στην κορυφή του κώδικα προσθέσαμε την εντολή <% layout('boilerplate') %> και με αυτό τον τρόπο όλες οι δηλώσεις που κάνουμε στο head μέρος του html γίνονται μία φορά και οποιοδήποτε κομμάτι κώδικα που θέλουμε να επαναλαμβάνεται προστίθεται στο αρχείο boilerplate.ejs.

Έπειτα έπρεπε να δημιουργήσουμε μια μπάρα πλοήγησης. Για να το κάνουμε αυτό, χρησιμοποιήσαμε το bootstrap που αναφέραμε παραπάνω. Αρχικά προσθέσαμε τις απαραίτητες γραμμές κώδικα στο boilerplate αρχείο έτσι ώστε να έχουμε πρόσβαση στις κλάσεις του framework σε όλες τις σελίδες της εφαρμογής μας. Έπειτα δημιουργήσαμε τη μπάρα πλοήγησης που θέλουμε βασιζόμενοι στην ιστοσελίδα του bootstrap και τα συστατικά που προσφέρει. Προαιρετικά και για καλύτερη δομή, μεταφέραμε τον κώδικα της μπάρας σε άλλο αρχείο που ονομάσαμε navbar.ejs και το συμπεριλάβαμε στο boilerplate χρησιμοποιώντας την εντολή <%- include('navbar') %>. Με αντίστοιχο τρόπο δημιουργήσαμε και το footer, το κάτω μέρος της σελίδας που αργότερα θα προσθέσουμε επιπλέον στοιχεία.

Για τη σελίδα index.ejs, τη σελίδα εμφάνισης όλων των εστιατορίων, χρησιμοποιήσαμε το grid system που προσφέρει το bootstrap το οποίο μας επιτρέπει να στοιχίσουμε τα αντικείμενα μας με τον τρόπο που θέλουμε. Επίσης χρησιμοποιήσαμε bootstrap cards για την εμφάνιση του κάθε εστιατορίου και με τις κατάλληλες κλάσεις τις πλάσαμε για να φτιάξουμε το επιθυμητό αποτέλεσμα.

Το ίδιο κάναμε και για τις σελίδες new, edit και show. Για τις σελίδες new και edit, η bootstrap προσφέρει έτοιμες κλάσεις για δημιουργία και στοίχιση φόρμας ενώ για τη show σελίδα δημιουργήσαμε μια κάρτα που περιλαμβάνει όλα τα στοιχεία του εστιατορίου. Επίσης, μέσω της bootstrap προσθέσαμε στις φόρμες μας στυλ για μια πρώτου επιπέδου επικύρωση δεδομένων πριν ο χρήστης υποβάλλει τη φόρμα.

4.3 Διαχείριση σφαλμάτων

Κατά τη διάρκεια ανάπτυξης ή χρήσης της εφαρμογής, συναντάμε διάφορα σφάλματα τα οποία μπορεί να προέρχονται από πολλές διαφορετικές πηγές όπως σύνδεση με τη βάση δεδομένων, μη ορθή χρήση κάποιου εργαλείου και άλλα. Αν αυτά τα σφάλματα μείνουν ανεξέλεγκτα, θα οδηγήσουν σε μια κακή εμπειρία χρήσης ή ακόμα και διάλυση της εφαρμογής. Η Express διαθέτει προκαθορισμένη λειτουργία διαχείρισης σφαλμάτων έτσι ώστε όταν κάνουμε αίτηση για ένα route το οποίο έχει πρόβλημα, θα επιστρέφεται το κατάλληλο status code και το μήνυμα περιγραφής τους σφάλματος στην οθόνη μας.

Πέρα από την προκαθορισμένη διαχείριση σφαλμάτων της Express, μπορούμε να δημιουργήσουμε και δικές μας συναρτήσεις για να διαχειριζόμαστε σφάλματα, το οποίο και κάναμε. Οι συναρτήσεις αυτές λειτουργούν ως ενδιάμεσο λογισμικό (middleware) που εκτελούνται τη στιγμή που ο server λαμβάνει το request και μέχρι να σταλθεί στον client το response. Ο λόγος για τον οποίο γίνεται αυτό είναι για να υπάρχει ξεχωριστή διαχείριση για κάθε είδους σφάλμα με το κατάλληλο μήνυμα και κωδικό. Επίσης προσθέσαμε στυλ στο μήνυμα μέσω της bootstrap για να αποκρύψουμε ή εμφανίσουμε τις λεπτομέρειες του σφάλματος ανάλογα με το που το συναντάμε. Εφόσον ετοιμάσαμε τη συνάρτηση σφάλματος που θέλουμε να χρησιμοποιήσουμε (catchAsync), την προσθέσαμε στα routes που μπορεί να συναντήσουμε σφάλμα και δημιουργήσαμε δικούς μας ελέγχους σφαλμάτων με κατάλληλο μήνυμα και κωδικό κατάστασης.

4.4 Προσθήκη αξιολογήσεων

Για την προσθήκη αξιολογήσεων, δημιουργήσαμε το review model όπου περιέχει βαθμολογία και κείμενο και ενημερώσαμε το restaurant model με την κατάλληλη αναφορά σε αυτό. Έτσι κάθε restaurant μπορεί να δεχθεί έναν πίνακα από reviews. Στη σελίδα προβολής ενός εστιατορίου προσθέσαμε τη φόρμα υποβολής καινούργιας κριτικής, ακριβώς κάτω από την εμφάνιση των στοιχείων του. Ενημερώσαμε το app.js με το route στο οποίο υποβάλει η φόρμα και μοιάζει κάπως έτσι:

```
app.post('/restaurants/:id/reviews', async(req, res) => {
   const restaurant = await Restaurant.findById(req.params.id);
   const review = new review(req.body.review);
   restaurant.reviews.push(review);
   await review.save();
   await restaurant.save();
   res.redirect('/restaurants/${restaurant._id}');
})
```

Αρχικά βρίσκουμε το εστιατόριο στο οποίο θέλουμε να προσθέσουμε την αξιολόγηση. Δημιουργούμε την αξιολόγηση. Προσθέτουμε την αξιολόγηση στον πίνακα των αξιολογήσεων του εστιατορίου. Αποθηκεύουμε αξιολόγηση και εστιατόριο και έπειτα ανακατευθύνουμε στη σελίδα του εστιατορίου στο οποίο βρισκόμαστε ήδη.

Έπειτα δημιουργήσαμε σχήμα και middleware συνάρτηση για επικύρωση των δεδομένων πριν την διαδικασία προσθήκης των αξιολογήσεων, όπως ακριβώς κάναμε και για τη δημιουργία εστιατορίου. Τα δεδομένα που θέλουμε να βρίσκονται κατά την υποβολή είναι το review object με υποχρεωτική βαθμολογία από το 1 ως το 5 και υποχρεωτικό σχόλιο.

```
reviewSchema = Joi.object({
   review: Joi.object({
      rating: Joi.number().required().min(1).max(5),
      body: Joi.string().required()
   }).required()
})
```

Ενημερώσαμε τη σελίδα εμφάνισης εστιατορίου ώστε να απεικονίζει τη λίστα με τις αξιολογήσεις που έχουν προστεθεί στο συγκεκριμένο εστιατόριο και προσθέσαμε βασικό στυλ και αποστάσεις χρησιμοποιώντας bootstrap. Κάτω από κάθε αξιολόγηση προσθέσαμε ένα κουμπί διαγραφής. Το route στο οποίο υποβάλλει το κουμπί, βρίσκει το εστιατόριο στο οποίο αναφερόμαστε, αφαιρεί από τον πίνακα αξιολογήσεων την αναφορά στην αξιολόγηση που θέλουμε να διαγράψουμε και φυσικά βρίσκει και διαγράφει την καταχώρηση της αξιολόγησης.

```
app.delete('/restaurants/:id/reviews/:reviewId', catchAsync(async (req,res)=>{
    const {id, reviewId } = req.params;
    await Restaurant.findByIdAndUpdate(id, {$pull: {reviews: reviewId}});
    await Review.findByIdAndDelete(reviewId);
    res.redirect(`/restaurants/${id}`);
}))
```

Τέλος, όταν διαγράφουμε ένα εστιατόριο πρέπει να διαγράφονται και οι αποθηκευμένες αξιολογήσεις σε αυτό. Η mongoose έχει δικές της middleware συναρτήσεις που δηλώνονται στο επίπεδο δήλωσης των σχημάτων και εκτελούνται πριν ή μετά από κάποια ασύγχρονη ενέργεια. Στην περίπτωσή μας, δηλώσαμε μια συνάρτηση όπου τρέχει μετά τη διαγραφή του εστιατορίου και διαγράφει όλες τις καταχωρήσεις στον πίνακα reviews.

```
RestaurantSchema.post('findOneAndDelete', async function (doc) {
    if (doc) {
        await Review.deleteMany({
            _id: {
                $in: doc.reviews
            }
        })
    }
})
```

4.5 Αυθεντικοποίηση χρηστών

4.5.1 Συνεδρίες

Η ιστοσελίδα δεν γνωρίζει ποιος χρήστης την επισκέπτεται κάθε φορά επειδή το HTTP πρωτόκολλο δεν διατηρεί καταστάσεις. Για να παρακάμψουμε αυτό τον περιορισμό και βασικό απαραίτητο για την προσθήκη λειτουργικότητας αυθεντικοποίησης χρηστών, είναι η προσθήκη συνεδριών (sessions). Ένα session χρησιμοποιείται για να αποθηκεύσει πληροφορίες στο server ώστε να μπορούν αξιοποιηθούν από τις διάφορες σελίδες ενός ιστότοπου. Οι πληροφορίες ξεκινάνε να ρέουν τη στιγμή που ο χρήστης εισέρχεται στην ιστοσελίδα και σταματάνε όταν εξέρχεται.

Όταν ο χρήστης κάνει επιτυχής σύνδεση, ένα μοναδικό session id με αυτές τις πληροφορίες αποθηκεύεται στη βάση δεδομένων και ταυτόχρονα επιστρέφεται πίσω στον browser σε μορφή cookie. Έτσι, για όλες τις μετέπειτα αιτήσεις σε αυτή τη διεύθυνση, ο browser αντιστοιχίζει το

session id που κατέχει με το session id της βάσης δεδομένων και γνωρίζει σε ποιον χρήστη αναφερόμαστε ώστε να απεικονίζει το κατάλληλα διαμορφωμένο περιεχόμενο για εκείνον.

Τα cookies που αναφέραμε παραπάνω είναι μικρά αρχεία κειμένου που μπορούν να δημιουργηθούν όταν επισκεπτόμαστε μια ιστοσελίδα και αποθηκεύονται τοπικά στον υπολογιστή μας. Το μέγιστο μέγεθός τους δεν ξεπερνάει τα 4KB και οι πληροφορίες που χωράνε σε αυτά είναι περιορισμένες.

Για να προσθέσουμε sessions στην εφαρμογή μας, εγκαταστήσαμε το κατάλληλο πακέτο μέσω του npm [12] και το προσθέσαμε στο app.js αρχείο. Μια ενδιάμεση συνάρτηση (app.use) στην αρχή του κώδικα αναλαμβάνει να το τρέχει κάθε φορά που κάνουμε αίτηση για οποιαδήποτε σελίδα της εφαρμογής.

4.5.2 Passport

Το Passport [13] είναι μια βιβλιοθήκη που μας βοηθάει να προσθέτουμε αυθεντικοποίηση σε διαδικτυακές εφαρμογές φτιαγμένες σε Express με τη χρήση ενδιάμεσης συνάρτησης. Αφού εγκαταστήσαμε τα κατάλληλα πακέτα passport [14] και passport-local [15] μέσω του npm, δημιουργήσαμε το user model, όπως ακριβώς κάναμε με το restaurant και το review. Το σχήμα στο user model περιέχει μόνο το email αφού το passport αναλαμβάνει και προσθέτει username και password. Στο app.js προσθέσαμε συναρτήσεις για να το αρχικοποιήσουμε και ορίσαμε τον τρόπο με τον οποίο θα αποθηκεύεται ο κάθε χρήστης στο τρέχον session. Επίσης προσθέσαμε τα κατάλληλα routes για εμφάνιση φόρμας εγγραφής, εγγραφή, εμφάνιση φόρμας σύνδεσης, σύνδεση και αποσύνδεση και τα αντίστοιχα αρχεία για τις φόρμες. Για την επικύρωση των δεδομένων κατά την εγγραφή χρησιμοποιήσαμε ένα πακέτο που ονομάζεται express-validator το οποίο σιγουρεύει ότι το πλήθος χαρακτήρων για τις καταχωρήσεις είναι επαρκές. Φροντίσαμε όταν γίνεται εγγραφή να παραμένει συνδεδεμένος ο χρήστης για να μη χρειαστεί να συνδεθεί εκ νέου και δημιουργήσαμε μια ενδιάμεση συνάρτηση με τη βοήθεια μιας μεθόδου του passport όπου κάνει έλεγχο για το αν έχει γίνει σύνδεση στη σελίδα ώστε να αποκρύψουμε συγκεκριμένες λειτουργίες σε περίπτωση που δεν έχει γίνει και να εμφανίσουμε κατάλληλο μήνυμα. Το μήνυμα εμφανίζεται με τη βοήθεια ενός πακέτου που ονομάζεται flash, παρέχεται από το npm [16] και μας επιτρέπει να περνάμε ορίσματα για την εμφάνιση μηνύματος επιτυχίας ή αποτυχίας κατά την ανακατεύθυνση σε άλλη σελίδα.

```
isLoggedIn = (req, res, next) => {
    if (!req.isAuthenticated()) {
        req.flash('error', 'You must be logged in!');
        return res.redirect('/login');
    }
    next();
}
```

Τέλος, ενημερώσαμε τη μπάρα πλοήγησης ώστε να περιέχει τους συνδέσμους για εγγραφή, σύνδεση και αποσύνδεση και με μια δομή ελέγχου φροντίσαμε να εμφανίζονται κατάλληλα ανάλογα με την κατάσταση χρήσης.

4.5.3 Εξουσιοδότηση χρήστη

Έχοντας ολοκληρώσει και τη λειτουργικότητα για τους χρήστες, το επόμενο που έπρεπε να κάνουμε ήταν να συσχετίσουμε χρήστες με εστιατόρια και αξιολογήσεις. Με αυτό τον τρόπο, μόνο ο ιδιοκτήτης κάθε εστιατορίου θα μπορεί να ενημερώσει ή να διαγράψει την καταχώρηση του. Αντίστοιχα με τις αξιολογήσεις, μόνο ο χρήστης που τη δημιούργησε θα έχει το δικαίωμα να τη διαγράψει.

Για τα εστιατόρια, ο τρόπος με τον οποίο το υλοποιήσαμε ήταν ενημερώνοντας το σχήμα στο μοντέλου του εστιατορίου με το πεδίο του ιδιοκτήτη και προσθέτοντας το user id σε αυτό το πεδίο κάθε φορά που δημιουργούμε ένα εστιατόριο. Έπειτα, με μια απλή δομή ελέγχου μεταξύ του συνδεδεμένου χρήστη και του ιδιοκτήτη του εστιατορίου μπορούμε να αποκρύψουμε ή να εμφανίσουμε τα πεδία για ενημέρωση ή διαγραφή. Επίσης δημιουργήσαμε μια ενδιάμεση συνάρτηση για να προστατέψουμε και τις αιτήσεις για τα αντίστοιχα routes.

```
isAuthor = async (req, res, next) => {
    const { id } = req.params;
    const restaurant = await Restaurant.findById(id);
    if (!restaurant.author.equals(req.user._id)) {
        req.flash('error', 'You do not have permission to do that!');
        return res.redirect(`/restaurants/${id}`);
    }
    next();
}
```

Την ίδια ακριβώς διαδικασία ακολουθήσαμε και για τη συσχέτιση αξιολογήσεων και χρηστών.

4.6 Διαμόρφωση κώδικα

Μέχρι τώρα, όλα τα route που δημιουργήσαμε βρίσκονταν στο app.js αρχείο. Μια καλή πρακτική είναι να τα μεταφέρουμε σε ξεχωριστά αρχεία και να τα ομαδοποιήσουμε με τη βοήθεια του Express Router. Το Express Router είναι μια κλάση που μας βοηθάει να δημιουργούμε χειριστές για τα routes μας σε μορφή ενδιάμεσων συναρτήσεων ώστε να έχουμε δομημένο και ευανάγνωστο κώδικα. Ο φάκελος που φιλοξενεί αυτά τα αρχεία ονομάζεται routes.

```
app.use('/restaurants', restaurantRoutes);
app.use('/restaurants/:id/reviews', reviewRoutes);
app.use('/', userRoutes);
```

Για να βοηθήσουμε περεταίρω με τη δομή του κώδικα, μεταφέραμε και όλη τη λειτουργικότητα από τη δήλωση των routes σε ξεχωριστά αρχεία δημιουργώντας συναρτήσεις που εξάγονται από αυτά και έπειτα εισάγονται στη δήλωση των route με το όνομά τους. Ο φάκελος που περιέχει αυτά τα αρχεία θα ονομάζεται controllers.

4.7 Προσθήκη φωτογραφιών

Επόμενος στόχος είναι να προσθέσουμε λειτουργικότητα ώστε να μπορεί ο χρήστης κατά την δημιουργία νέου εστιατορίου να ανεβάσει φωτογραφίες από τον υπολογιστή του που το αντιπροσωπεύουν. Ο τρόπος με τον οποίο το υλοποιήσαμε είναι αποθηκεύοντας τις φωτογραφίες σε μια εξωτερική πηγή από την οποία θα αντλούμε τη διεύθυνση φιλοξενίας και θα αποθηκεύουμε στη βάση δεδομένων μας.

Αρχικά αλλάξαμε το encryption type (τύπος κρυπτογράφησης) της φόρμας δημιουργίας νέου εστιατορίου σε multipart/form-data για να μπορεί να περιέχει εισαγωγή αρχείου και προσθέσαμε το κατάλληλο στοιχείο. Για να προσπελάσουμε τώρα την καινούργια φόρμα χρειάστηκε να εγκαταστήσουμε το Multer [17]. Το Multer είναι μια ακόμα ενδιάμεση συνάρτηση του Node.js, προσφέρεται μέσω του npm και μας επιτρέπει να διαχειριζόμαστε φόρμες που περιέχουν αρχεία. Η εξωτερική πηγή που χρησιμοποιήσαμε για αποθήκευση των φωτογραφιών ονομάζεται Cloudinary [18] η οποία προσφέρει υπηρεσίες χωρίς αλλά και με χρέωση για το ανέβασμα, την αποθήκευση και τον χειρισμό αρχείων εικόνας και βίντεο. Εμείς φυσικά χρησιμοποιούμε το πακέτο χωρίς χρέωση.

Για να συνδέσουμε το Cloudinary με την εφαρμογή, χρειάστηκε να εγκαταστήσουμε δύο ακόμα πακέτα από το npm, το cloudinary [19] και το multer-storage-cloudinary [20]. Κάναμε τις απαραίτητες ρυθμίσεις παραμέτρων ώστε να συνδέσουμε το Cloudinary με την εφαρμογή και ορίσαμε τους επιτρεπτούς τύπους αρχείων για ανέβασμα στο φάκελο που θέλουμε να γίνεται η αποθήκευση. Μετά χρησιμοποιήσαμε την ενδιάμεση συνάρτηση του multer για ανέβασμα στους server του Cloudinary στο route όπου δημιουργείται το εστιατόριο. Από την αίτηση που κάνουμε, παίρνουμε πίσω το όνομα του αρχείου και τη διεύθυνση στην οποία φιλοξενείται. Στη συνέχεια προσθέσαμε στο μοντέλου του εστιατορίου το πεδίο images το οποίο είναι ένας πίνακας από objects με διεύθυνση και όνομα. Ο λόγος που ορίσαμε πίνακα είναι γιατί επιτρέπεται και το ανέβασμα πολλών φωτογραφιών. Μετά ενημερώσαμε τον κώδικα στο φάκελο controllers ώστε να αποθηκεύει αυτά τα δεδομένα στην Mongo.

Τώρα πρέπει να εμφανίσουμε τις αποθηκευμένες φωτογραφίες. Στην show.ejs σελίδα, εντός της κάρτας εμφάνισης εστιατορίου, προσθέσαμε ένα στοιχείο τύπου carousel που παρέχει η bootstrap ώστε να εμφανίζει κυκλικά τις φωτογραφίες. Το ίδιο κάναμε και στην index.ejs σελίδα με τη διαφορά ότι δε προσθέσαμε κυκλική εμφάνιση. Η στατική εμφάνιση του πρώτου στοιχείου του πίνακα αποθηκευμένων φωτογραφιών αρκεί. Επίσης το Cloudinary μας παρέχει ένα πολύ εύχρηστο API μεταμόρφωσης εικόνας. API (Application Programming Interface) ονομάζεται το λογισμικό που επιτρέπει σε δύο εφαρμογές να επικοινωνούν μεταξύ τους. Με τη βοήθεια αυτού του εργαλείου και επειδή κάθε εικόνα έχει τις δικές της διαστάσεις, θα δημιουργήσουμε μια εικονική μεταβλητή στο μοντέλου του εστιατορίου ώστε κάθε φορά να γίνεται εμφάνιση της εικόνας με τον τρόπο που θέλουμε εμείς. Οι εικονικές μεταβλητές λειτουργούν σαν τις κανονικές και μας βοηθάνε όταν θέλουμε να χρησιμοποιήσουμε κάτι με αναφορά από τη βάση χωρίς να ξοδέψουμε επιπλέον χώρο αποθήκευσης.

Τέλος, για τη διαγραφή φωτογραφιών, στο edit.ejs αρχείο κάνουμε εμφάνιση των αποθηκευμένων φωτογραφιών κάθε εστιατορίου σε μικρογραφίες χρησιμοποιώντας εικονικές μεταβλητές και δίπλα τους υπάρχει ένα πλαίσιο ελέγχου που μπορούμε να τσεκάρουμε αν θέλουμε να τις διαγράψουμε. Στο controller αρχείο που διαχειρίζεται το route της φόρμας ενημέρωσης, λαμβάνουμε το αίτημα, ελέγχουμε αν υπάρχει όρισμα για διαγραφή φωτογραφιών και καλούμε την κατάλληλη συνάρτηση του multer για διαγραφή από το Cloudinary. Επίσης διαγράφουμε και την καταχώρησή της στη Mongo.

4.8 Προσθήκη χάρτη

Το τελευταίο σημαντικό χαρακτηριστικό που προσθέσαμε στην εφαρμογή μας είναι η παροχή χαρτών. Θέλουμε στη σελίδα εμφάνισης κάθε εστιατορίου να εμφανίζεται ένας χάρτης με σημαδεμένη πάνω σε αυτόν την τοποθεσία του. Επίσης στη σελίδα εμφάνισης όλων των εστιατορίων θέλουμε να εμφανίζεται ένας παγκόσμιος χάρτης που θα περιέχει σημαδεμένες τις τοποθεσίες όλων των αποθηκευμένων εστιατορίων. Για την εφαρμογή χαρτών χρησιμοποιήσαμε το Mapbox [21]. Το Mapbox είναι μια δημοφιλής πλατφόρμα παροχής υπηρεσιών τοποθεσίας με πολλές δυνατότητες οι οποίες στην πλειοψηφία τους προσφέρονται χωρίς χρέωση.

Αρχικά ενημερώσαμε το restaurant model με πεδία για διεύθυνση, ταχυδρομικό κώδικα, πόλη και χώρα. Έπειτα ενημερώσαμε τις φόρμες για δημιουργία και ενημέρωση εστιατορίων ώστε να δέχονται αυτά τα πεδία και αντίστοιχα τα εμφανίζουμε στη σελίδα εμφάνισης όλων των εστιατορίων (index.ejs) και στη σελίδα προβολής περισσότερων πληροφοριών για ένα εστιατόριο (show.ejs). Επίσης, αφού κάναμε εγγραφή στη σελίδα του mapbox, λάβαμε ένα κλειδί πρόσβασης (API access token) το οποίο έπρεπε να συμπεριλάβουμε στον κώδικά μας ώστε να έχουμε πρόσβαση στις υπηρεσίες τοποθεσίας.

Ο τρόπος με τον οποίο μια τοποθεσία σημαδεύεται στο χάρτη είναι εισάγοντας τις συντεταγμένες της. Ο χρήστης προφανώς δεν έχει την υποχρέωση να γνωρίζει κάτι τέτοιο οπότε χρησιμοποιήσαμε μια τεχνική που ονομάζεται γεωκωδικοποίηση (geocoding). Η γεωκωδικοποίηση είναι η διαδικασία που μετατρέπει τη διεύθυνση μιας τοποθεσίας σε συντεταγμένες, δηλαδή σε γεωγραφικό μήκος και γεωγραφικό πλάτος. Το mapbox προσφέρει api client για Node που αναλαμβάνει αυτή και άλλες λειτουργίες. Κάναμε εγκατάσταση το πακέτο @mapbox/mapbox-sdk [22] από το npm. Ο τρόπος με τον οποίο το χρησιμοποιήσαμε είναι δημιουργώντας τον client, δημιουργώντας το αίτημα και μετά στέλνοντας το αίτημα.

```
const mbxGeocoding = require("@mapbox/mapbox-sdk/services/geocoding");
const geocoder = mbxGeocoding({ accessToken: mapBoxToken });

const geoData = await geocoder.forwardGeocode({
      query: req.body.restaurant.address + ', ' + req.body.restaurant.postcode + ', ' +
     req.body.restaurant.city + ', ' + req.body.restaurant.country,
      limit: 1,
}).send()
```

Για να αποθηκεύσουμε την απάντηση από το παραπάνω αίτημα, έπρεπε να ενημερώσουμε το μοντέλο του εστιατορίου με κατάλληλο τρόπο που υποδεικνύει η mongoose για

αποθήκευση δεδομένων τύπου τοποθεσίας και μετά στο controller αρχείο για τη δημιουργία εστιατορίου επισυνάψαμε την απάντηση του αιτήματος στο πεδίο του εστιατορίου που δημιουργήσαμε. Πλέον έχουμε τις συντεταγμένες για το κάθε εστιατόριο στη βάση δεδομένων.

Για την εμφάνιση του χάρτη χρησιμοποιήσαμε μια Javascript βιβλιοθήκη του mapbox, την Mapbox GL JS, που χρησιμοποιεί WebGL για να αποδώσει διαδραστικούς χάρτες στο στυλ του mapbox. Το WebGL (Web Graphics Library) είναι ένα Javascript API που χρησιμοποιείται για τη δημιουργία δισδιάστατων και τρισδιάστατων γραφικών σε έναν browser. Απαραίτητο για τη χρήση της, ήταν να προσθέσουμε τα αναγκαία CSS και Javascript αρχεία στο head μέρος της εφαρμογής μας. Μετά, συμβουλευόμενοι το documentation της βιβλιοθήκης, εισάγαμε εντός script tags στο τέλος της σελίδας που θέλουμε να κάνουμε εμφάνιση του χάρτη, τον κώδικα ορισμού του και προσθέσαμε ένα div με id=”map” στο σημείο που θέλουμε να εμφανιστεί. Αφού ο χάρτης εμφανίζεται κανονικά, περάσαμε το restaurant object στο client μέρος για να έχουμε πρόσβαση στα στοιχεία του. Κεντράραμε το χάρτη για να φαίνεται σωστά μεγεθυμένος στην περιοχή των συντεταγμένων του εστιατορίου. Στη συνέχεια, στον κώδικα ορισμού του χάρτη, δημιουργήσαμε ένα mapbox σημάδι (marker) με τις συντεταγμένες και ένα μήνυμα προβολής σε αναδυόμενο παράθυρο και τα προσθέσαμε.

```
new mapboxgl.Marker()
    .setLngLat(restaurant.geometry.coordinates)
    .setPopup(
        new mapboxgl.Popup({ offset: 25 })
            .setHTML(
                `<h4>${restaurant.title}</h4><p>${restaurant.location}</p>`
            )
    )
    .addTo(map);
```

Ο δεύτερος χάρτης που δημιουργήσαμε θέλαμε να περιέχει σημάδια για κάθε εστιατόριο που βρίσκεται αποθηκευμένο στην εφαρμογή μας και να τα ομαδοποιεί ανάλογα με την περιοχή στην οποία βρίσκονται. Πάλι, συμβουλευόμενοι το documentation που προσφέρει το mapbox, εισήγαμε τον κώδικα ορισμού του χάρτη τον οποίο θέλουμε να χρησιμοποιήσουμε στο τέλος της σελίδας εμφάνισης όλων των εστιατορίων. Περάσαμε πάλι τα restaurant objects στο client μέρος για να έχουμε πρόσβαση στα στοιχεία τους και τα αποθηκεύσαμε σε μορφή κατάλληλη για ανάγνωση από το mapbox. Παραμετροποιήσαμε τον κώδικα ορισμού ώστε να δέχεται δεδομένα από τα εστιατόρια και ρυθμίσαμε το στυλ εμφάνισής του. Επίσης προσθέσαμε αναδυόμενο παράθυρο με μήνυμα που περιέχει σύνδεσμο παραπομπής στο

εκάστοτε εστιατόριο όταν ο χρήστης κάνει κλικ σε κάποιο σημείο. Τέλος, προσθέσαμε το div με id=”map” στο σημείο της σελίδας που θέλουμε να εμφανίσουμε το χάρτη.

4.9 Επιπλέον στοιχεία

Έχοντας το βασικό κορμό της εφαρμογής έτοιμο, θέλαμε να προσθέσουμε μερικά επιπλέον στοιχεία ώστε η ιστοσελίδα να γίνει ευπαρουσίαστη και να ανταποκρίνεται σε περισσότερες απαιτήσεις της πραγματικής ζωής. Το πρώτο που προσθέσαμε είναι μια CSS βιβλιοθήκη για την προσθήκη και απεικόνιση της βαθμολογίας σε μορφή αστεριών. Η βιβλιοθήκη ονομάζεται Starability.css [23]. Για να τη χρησιμοποιήσουμε, δημιουργήσαμε ένα αρχείο stars.css και εισήγαμε τον κώδικά της μέσα σε αυτό. Έπειτα το προσθέσαμε στη σελίδα που θέλουμε να χρησιμοποιήσουμε αυτή την λειτουργία, δηλαδή στη σελίδα εμφάνισης εστιατορίου (show.ejs). Θέτοντας τις κατάλληλες κλάσεις στη φόρμα δημιουργίας νέων αξιολογήσεων και στην κάρτα προβολής αξιολογήσεων, πλέον η βαθμολογία εμφανίζεται σε μορφή αστεριών και ο χρήστης μπορεί να προσθέσει την βαθμολογία του επιλέγοντας τον αριθμό αστεριών στην οθόνη του. Επίσης, στο μοντέλο του εστιατορίου δημιουργήσαμε δύο εικονικές μεταβλητές που θα υπολογίζουν το πλήθος των αξιολογήσεων που έχουν προστεθεί και τη μέση τιμή αυτών. Τις δύο αυτές βοηθητικές μεταβλητές τις χρησιμοποιήσαμε για να εμφανίσουμε το μέσο όρο των βαθμολογιών στον τίτλο του εστιατορίου σε μορφή αστεριών. Επειδή η βιβλιοθήκη που χρησιμοποιήσαμε δεν υποστηρίζει μισά αστέρια, δημιουργήσαμε δικές μας κλάσεις με βήμα 0,5 ώστε να μπορεί να εμφανίζεται σωστά ο μέσος όρος βαθμολογιών με υποδιαστολή. Επίσης, αφήνοντας τον κέρσορα πάνω από τη βαθμολογία μπορούμε να δούμε το πλήθος των αξιολογήσεων και τη βαθμολογία ολογράφως.

Στο μοντέλο των αξιολογήσεων προσθέσαμε το πεδίο date όπου αποθηκεύουμε την ημερομηνία και την ώρα την οποία δημιουργήθηκε. Η αποθήκευση γίνεται στο controller αρχείο κατά τη δημιουργία νέας κριτικής και η εμφάνισή τους γίνεται στην κάρτα προβολής αξιολογήσεων.

Στο μοντέλο του εστιατορίου προσθέσαμε επίσης μέγιστη τιμή δείπνου ανά άτομο, κουζίνες που υποστηρίζει το εστιατόριο, ιστοσελίδα, τηλέφωνο και email. Για τα αντίστοιχα πεδία δημιουργήσαμε ελέγχους εγκυρότητας τιμών και προστέθηκαν στις φόρμες δημιουργίας, ενημέρωσης καθώς και στις κάρτες εμφάνισης.

Στην αρχική σελίδα της εφαρμογής (home.ejs), δημιουργήσαμε μια μπάρα πλοήγησης στην κορυφή με παρόμοιο στυλ εμφάνισης όπως τις υπόλοιπες σελίδες. Προσθέσαμε μια εικόνα για

το παρασκήνιο, τον τίτλο της εφαρμογής και ένα μήνυμα καλωσορίσματος Επίσης προσθέσαμε μια μπάρα αναζήτησης για το χρήστη σε μορφή φόρμας. Η φόρμα υποβάλλει στη σελίδα εμφάνισης όλων των εστιατορίων όπου και γίνεται η εύρεση των εστιατορίων που ταιριάζουν με τον όρο αναζήτησης. Ο χρήστης μπορεί να αναζητήσει με βάση των τίτλο, την περιγραφή, τη διεύθυνση, τον ταχυδρομικό κώδικα, την πόλη, τη χώρα ή τον τύπο της κουζίνας που σερβίρεται. Οι καταχωρήσεις που ταιριάζουν με την αναζήτηση επιστρέφονται στην ίδια τη σελίδα εμφάνισης όλων των εστιατορίων, σημαδεύεται η τοποθεσία τους στο χάρτη και εμφανίζεται η λίστα προβολής τους. Τέλος, προσθέσαμε στυλ σε όλες τις σελίδες της εφαρμογής μέσω bootstrap για να έχουμε ένα δομημένο και όμορφο αποτέλεσμα.

4.10 Ασφάλεια

Η έννοια της ασφάλειας στο διαδίκτυο είναι η διαδικασία προστασίας μιας ιστοσελίδας ή εφαρμογής ενάντια σε διάφορες απειλές ή επιθέσεις που εκμεταλλεύονται αδυναμίες στον κώδικα τους. Μια τέτοια αδυναμία μπορεί να φέρει σε κίνδυνο τα προσωπικά δεδομένα των χρηστών ή ακόμα και να καταστρέψει ολοσχερώς την εφαρμογή. Παρ' όλο που υπάρχουν εταιρείες που ασχολούνται αποκλειστικά με την παροχή υπηρεσιών ασφάλειας σε διαδικτυακές εφαρμογές, είναι καθήκον του δημιουργού να την προστατέψει όσο καλύτερα μπορεί.

Το πρώτο ζήτημα που επιλύσαμε είναι το NoSQL Injection. Ένα site ευάλωτο σε τέτοιες επιθέσεις μπορεί να επιτρέψει στον κακόβουλο χρήστη να εισάγει κώδικα σε πεδία που έχουν άμεση επικοινωνία με τη βάση δεδομένων με σκοπό να αποσπάσει δεδομένα χρηστών ή να παρακάμψει σημεία αυθεντικοποίησης. Ο τρόπος με τον οποίο το αντιμετωπίσαμε είναι απαγορεύοντας στους χρήστες να εισάγουν ειδικούς χαρακτήρες που χρησιμοποιεί η Mongo στα πεδία εισόδου δεδομένων. Το πακέτο που χρησιμοποιήσαμε ονομάζεται express-mongo-sanitize [24] και μπορούμε να το εγκαταστήσουμε από το npm. Για να το εντάξουμε στην εφαρμογή μας, μια απλή κλήση μέσω ενδιάμεσης συνάρτησης πριν την δήλωση των routes αρκεί.

Μία ακόμα αδυναμία που καλούμαστε να αντιμετωπίσουμε είναι αυτή του Cross Site Scripting (XSS). Αυτή η αδυναμία επιτρέπει σε αυτόν που επιτίθεται να εισάγει στο θύμα κώδικα που θα τρέξει στον browser και θα εκτελέσει κάποια κακόβουλη ενέργεια. Για να δημιουργήσουμε την κατάλληλη άμυνα απέναντι σε αυτό, εγκαταστήσαμε από το npm ένα πακέτο που ονομάζεται sanitize-html [25] και αυτό το οποίο κάνει είναι να αφαιρεί κάθε

HTML tag από μια συμβολοσειρά. Μετά, με τη βοήθεια αυτού, δημιουργήσαμε μια προέκταση στο ήδη υπάρχον εργαλείο Joi που χρησιμοποιήσαμε για επικύρωση δεδομένων κατά την αποθήκευση στη βάση δεδομένων. Προσθέσαμε αυτή την προέκταση στο σχήμα των δεδομένων που έχουμε για αποθήκευση έτσι ώστε να μην επιτρέπεται η εισαγωγή οποιουδήποτε HTML tag σε αυτά τα πεδία.

Το τελευταίο εργαλείο προστασίας που προσθέσαμε στην εφαρμογή μας ονομάζεται Helmet [26]. Πρώτα το εγκαταστήσαμε από το npm και έπειτα το τρέχουμε ως ενδιάμεση συνάρτηση για την εφαρμογή μας. Με το helmet παρέχουμε την απαραίτητη προστασία στα HTTP headers που μέχρι τώρα δεν είχαν. Τα HTTP headers είναι ο πυρήνας των http request και response και περιέχουν σημαντικές πληροφορίες για τον server και τον client.

4.11 Πηγαίνοντας online

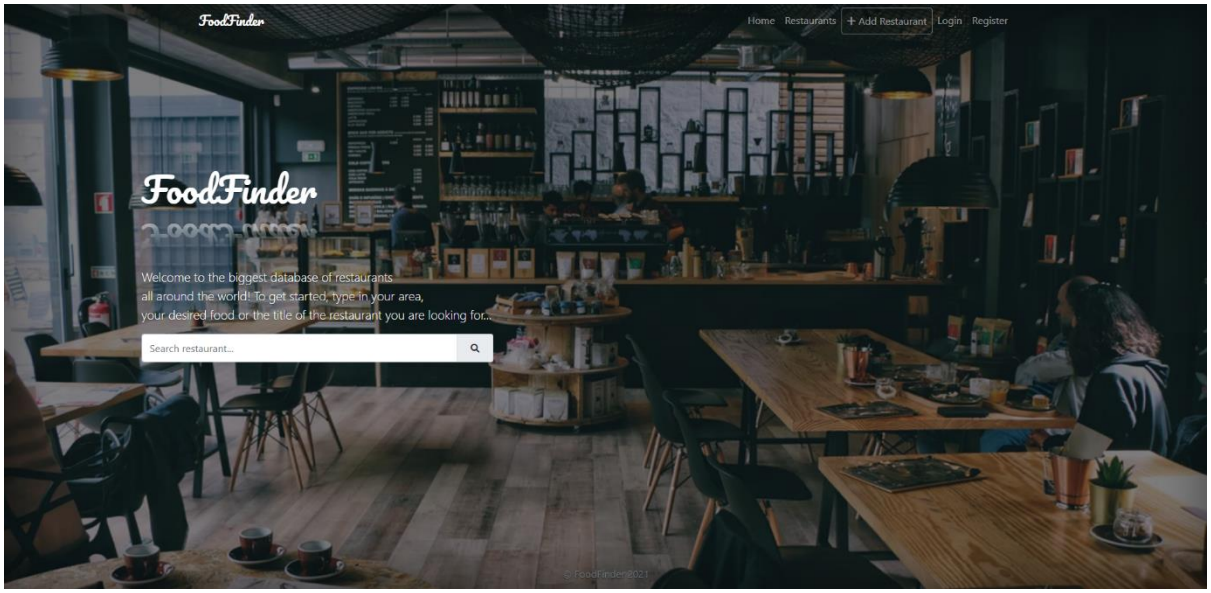
Καθ' όλη τη διάρκεια ανάπτυξης κώδικα και μέχρι τώρα, η εφαρμογή έτρεχε τοπικά όπως επίσης τοπικά ήταν ρυθμισμένη και η βάση δεδομένων μας. Αυτό σημαίνει ότι ο υπολογιστής μας είχε το ρόλο του server και στις δυο περιπτώσεις. Στόχος αυτού του κεφαλαίου είναι να κάνουμε την εφαρμογή μας να βρίσκεται online και να είναι λειτουργική και προσβάσιμη από όλους.

Υποθετικά θα μπορούσαμε να συνδέσουμε την τοπική βάση δεδομένων με την εφαρμογή μας όταν αυτή βρίσκεται online ωστόσο στον πραγματικό κόσμο είναι καλή πρακτική να υπάρχουν δύο βάσεις, μία για ανάπτυξη κώδικα και μία που θα φιλοξενεί πραγματικούς χρήστες. Αυτό γίνεται διότι όταν χρειαστεί να κάνουμε την οποιαδήποτε δοκιμή στην ιστοσελίδα, θέλουμε να είμαστε σίγουροι ότι τα πραγματικά δεδομένα των χρηστών μας είναι ασφαλή. Η υπηρεσία βάσης δεδομένων νέφους που χρησιμοποιήσαμε ονομάζεται MongoDB Atlas και είναι προϊόν της Mongo. Στην ουσία είναι η ίδια βάση που χρησιμοποιούσαμε μέχρι τώρα αλλά αντί να βρίσκεται στον υπολογιστή μας, βρίσκεται online. Κάναμε εγγραφή χωρίς χρέωση στην ιστοσελίδα του MongoDB Atlas, δημιουργήσαμε τη βάση και λάβαμε το url που πρέπει να εισάγουμε στον κώδικά μας για να γίνει η σύνδεση. Το url περιέχει τα στοιχεία του διαχειριστή οπότε πρέπει με κάποιο τρόπο να τα αποκρύψουμε. Χρησιμοποιήσαμε ένα πακέτο που εγκαταστήσαμε από το npm και ονομάζεται dotenv [27]. Με τη βοήθεια αυτού μπορούμε να δημιουργήσουμε κρυφές μεταβλητές περιβάλλοντος και να τις διαχωρίσουμε από τον κώδικά μας έχοντας όμως πρόσβαση σε αυτές. Ορίσαμε σε αυτό το αρχείο οποιαδήποτε τέτοια

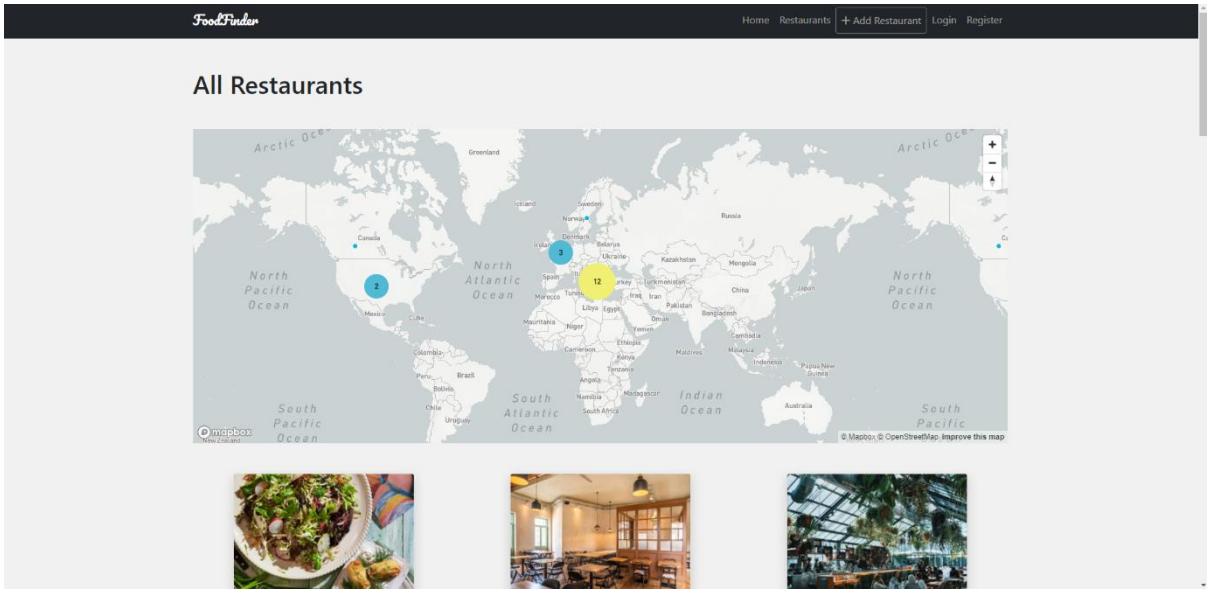
μεταβλητή θέλουμε. Μετά εισήγαμε τη μεταβλητή που περιέχει το url της βάσης στον κώδικά μας στο πεδίο που γίνεται σύνδεση. Πλέον η εφαρμογή μας εξυπηρετείται από την online βάση του MongoDB Atlas. Στη συνέχεια εγκαταστήσαμε το πακέτο connect-mongo [28] από το npm, ώστε να αποθηκεύουμε το εκάστοτε session στη βάση δεδομένων αντί για τη μνήμη.

Η πλατφόρμα που χρησιμοποιήσαμε για την φιλοξενία της εφαρμογής μας στο διαδίκτυο ονομάζεται Heroku [29]. Δημιουργήσαμε λογαριασμό χωρίς χρέωση και κατεβάσαμε το απαραίτητο εργαλείο ώστε να δημιουργήσουμε και να διαχειριστούμε την εφαρμογή από το terminal του υπολογιστή μας. Για να ανεβάσουμε την εφαρμογή μας, εγκαταστήσαμε και χρησιμοποιήσαμε το git [30] αλλά πριν από αυτό δημιουργήσαμε το αρχείο .gitignore και σημειώσαμε μέσα σε αυτό τα αρχεία που δε θέλουμε να ανέβουν (.env). Το git είναι ένα λογισμικό που μας βοηθάει στη διαχείριση του project μας. Δημιουργήσαμε χώρο για την εφαρμογή μας μέσα από το command line και έπειτα ανεβάσαμε τα αρχεία του κώδικα σε αυτόν με τη βοήθεια του git. Η εφαρμογή πλέον βρίσκεται online και είναι προσβάσιμη από όλους στην διεύθυνση: https://vast-shore-70286.herokuapp.com/

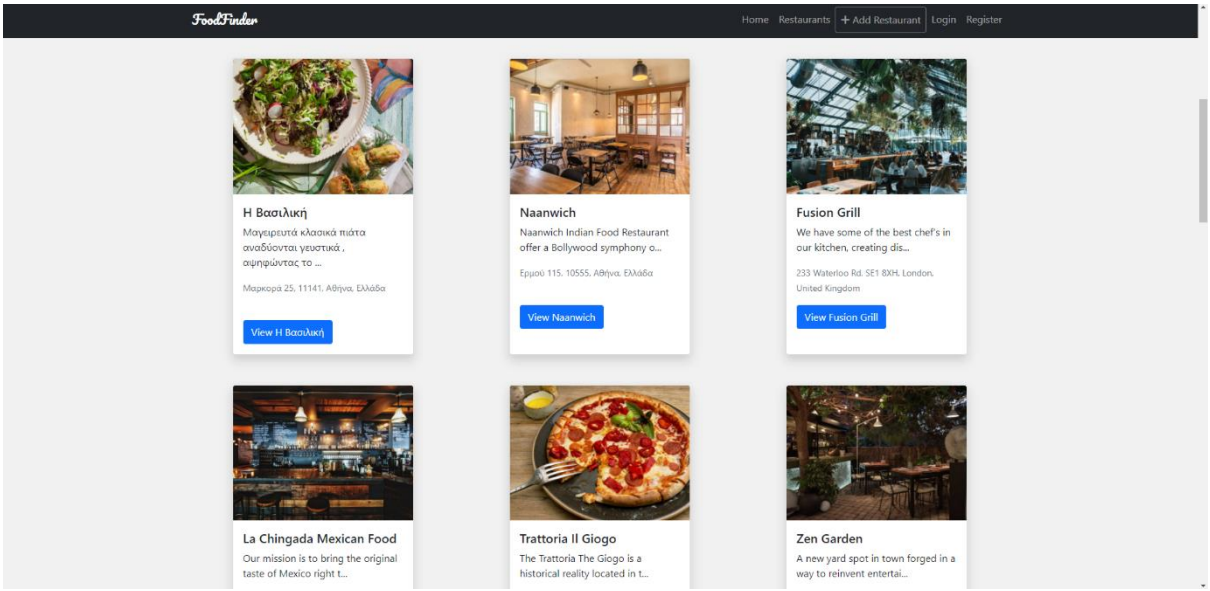
5. Τελικό αποτέλεσμα



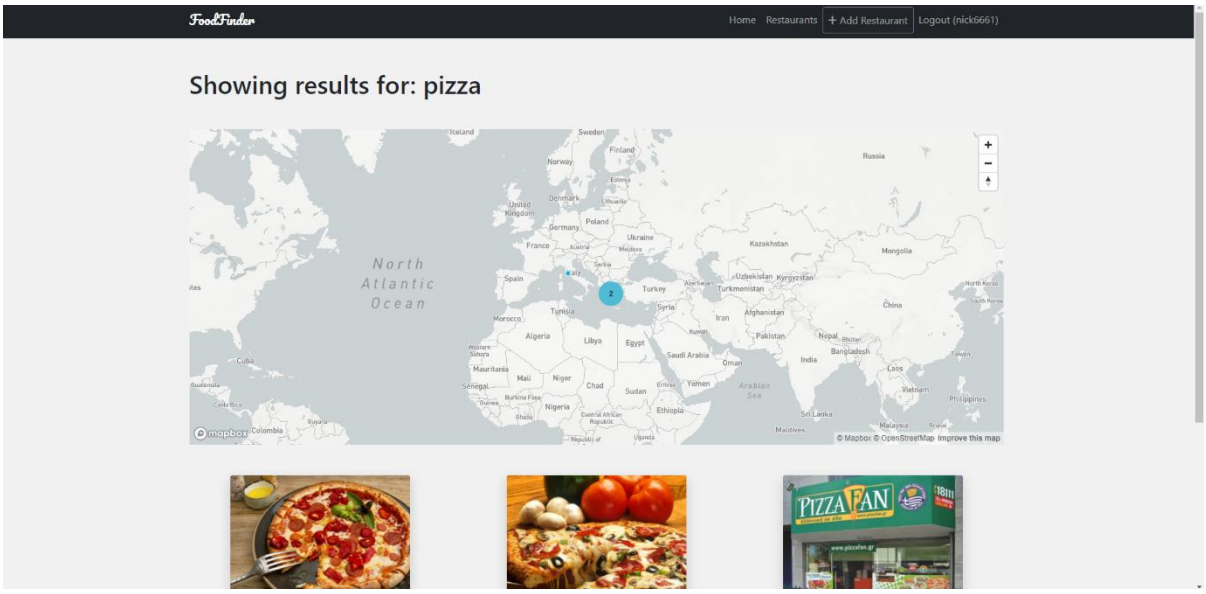
Εικόνα 4 - Αρχική σελίδα (Home)



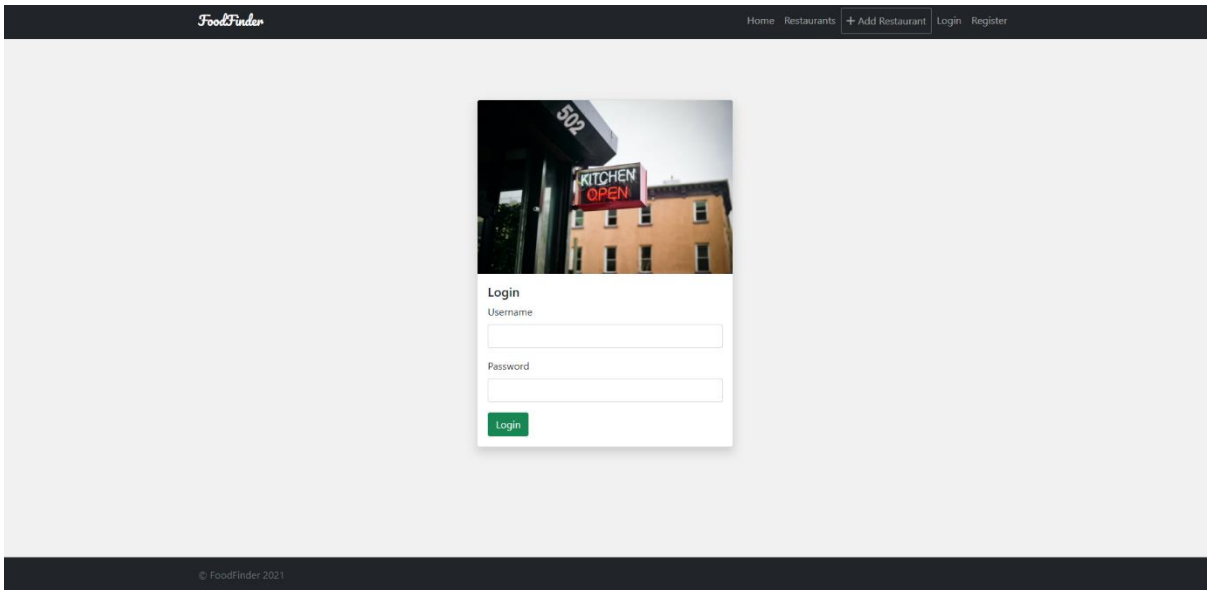
Εικόνα 5 - Σελίδα εμφάνισης όλων των εστιατορίων (Restaurants)



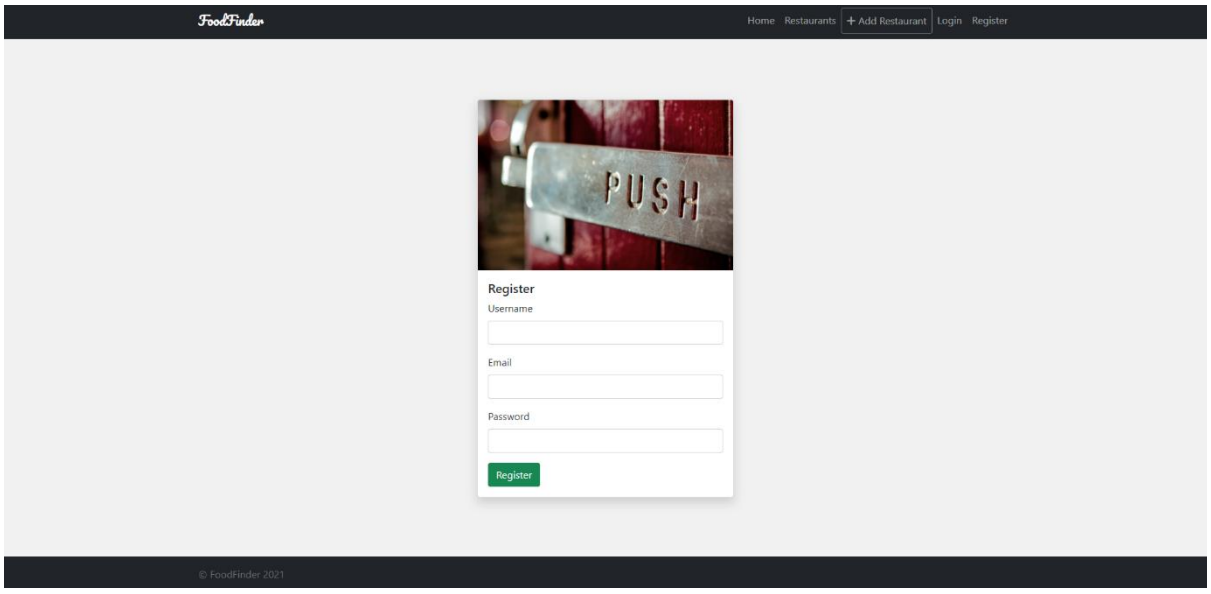
Εικόνα 6 - Σελίδα εμφάνισης όλων των εστιατορίων (Restaurants) (Συν.)



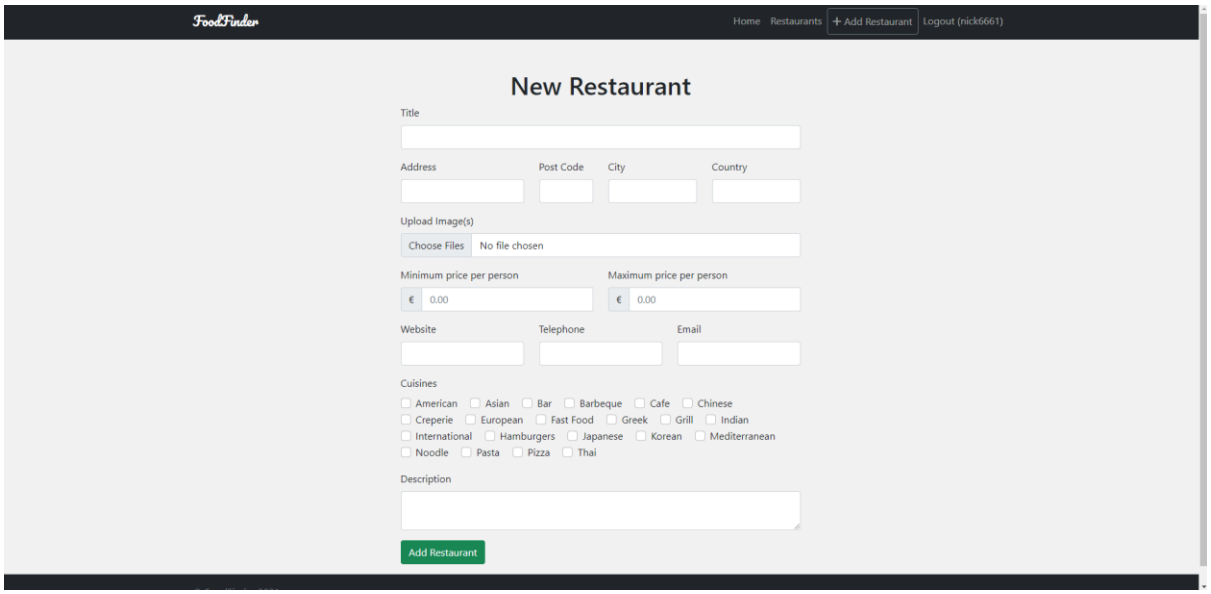
Εικόνα 7 - Σελίδα εμφάνισης όλων των εστιατορίων μετά από αναζήτηση (Restaurants)



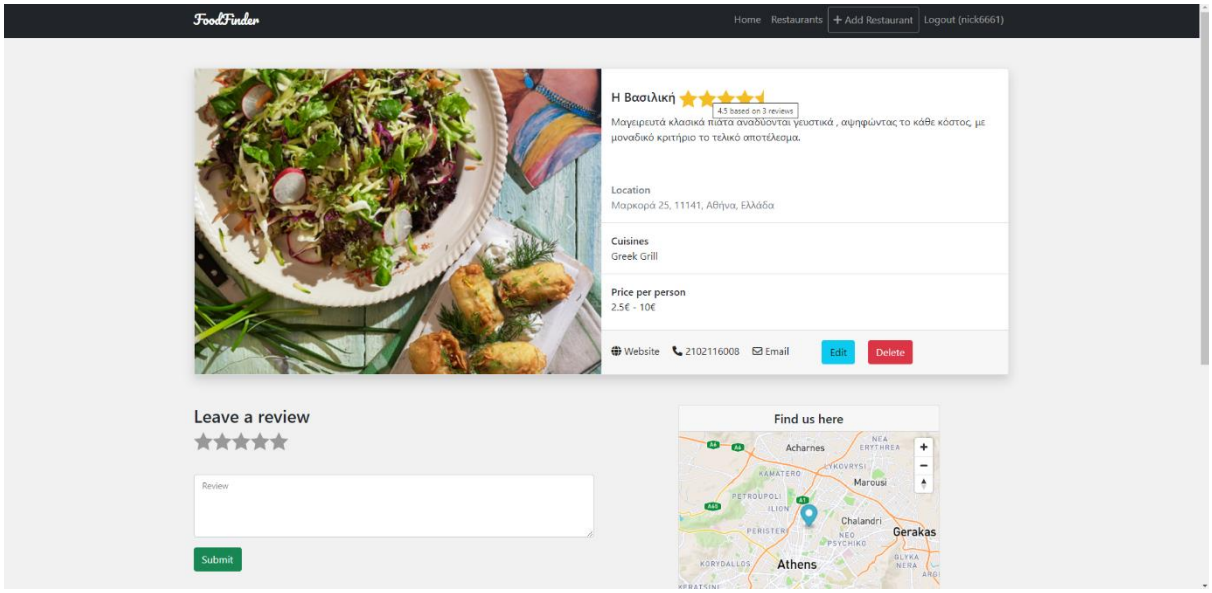
Εικόνα 8 - Σελίδα σύνδεσης χρήστη (Login)



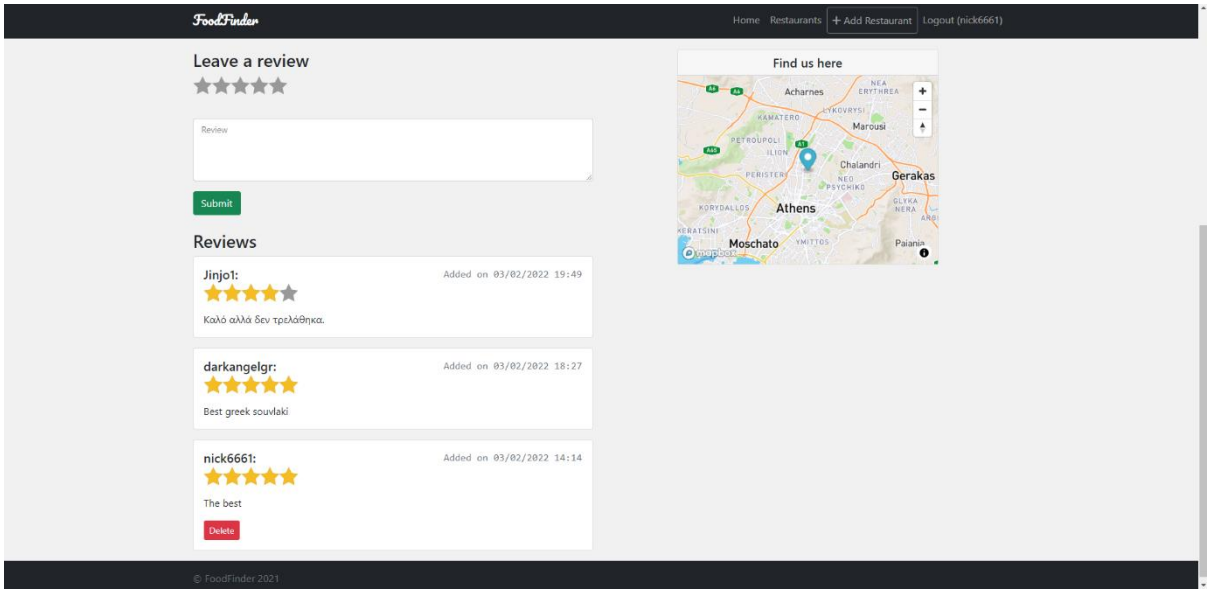
Εικόνα 9 - Σελίδα εγγραφής χρήστη (Register)



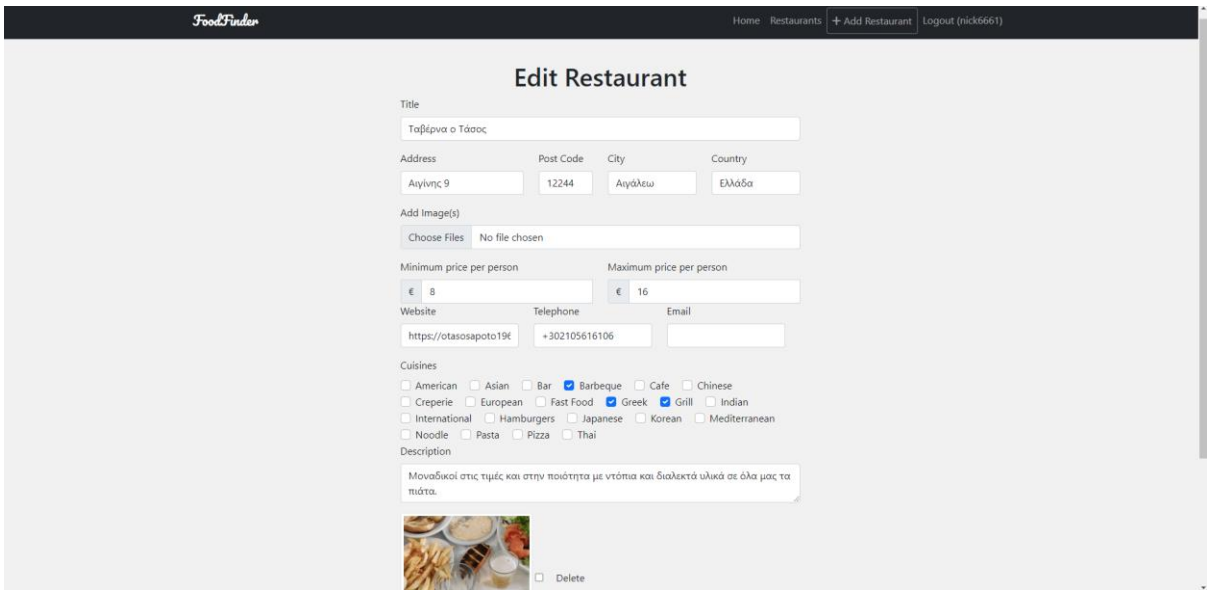
Εικόνα 10 - Σελίδα προσθήκης εστιατορίου (New)



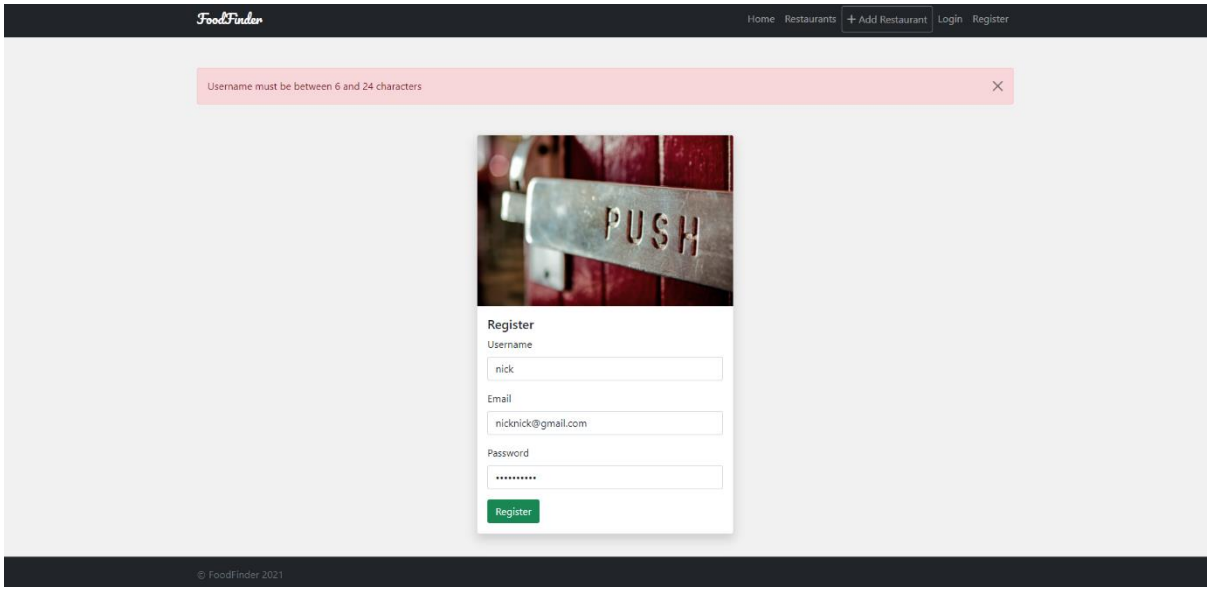
Εικόνα 11 - Σελίδα προβολής εστιατορίου (Ιδιοκτήτης καταχώρησης)



Εικόνα 12 - Σελίδα προβολής εστιατορίου (Ιδιοκτήτης καταχώρησης) (Συν.)



Εικόνα 13 - Σελίδα ενημέρωσης εστιατορίου



Εικόνα 14 - Μήνυμα σφάλματος κατά την εγγραφή

6. Συμπεράσματα και μελλοντικές επεκτάσεις

Στην παρούσα εργασία αναλύσαμε τον τρόπο με τον οποίο δημιουργείται μια διαδικτυακή εφαρμογή από το μηδέν μέχρι και το σημείο που βρίσκεται online. Είδαμε πως συνδυάζονται πολλές διαφορετικές γλώσσες, πακέτα και μοντέρνες τεχνολογίες. Εν τέλει, φτιάξαμε ένα website διαχείρισης εστιατορίων το οποίο είναι έτοιμο να φιλοξενήσει καταχωρήσεις από όλο τον κόσμο.

Η εφαρμογή βρίσκει άμεση χρησιμότητα στον πραγματικό κόσμο, ωστόσο πρέπει πρώτα να προστεθούν κάποια θεμελιώδη χαρακτηριστικά τα οποία δεν αναπτύχθηκαν κατά την εκπόνηση της εργασίας διότι δεν είχαν ιδιαίτερο προγραμματιστικό ενδιαφέρον. Μερικά από αυτά είναι:

- Προσθήκη προφίλ χρήστη
- Προσθήκη μενού διαχειριστή
- Αναφορά καταχωρήσεων
- Λεπτομερής διαχείριση σφαλμάτων
- Επαλήθευση εστιατορίου για τα δικαιώματα ιδιοκτησίας
- Ελκυστικό γραφικό περιβάλλον και υποστήριξη κάθε τύπου οθόνης

Με την προσθήκη των κατάλληλων χαρακτηριστικών, η εφαρμογή θα είναι έτοιμη να λειτουργήσει ομαλά και να αναπτυχθεί σε παγκόσμια κλίμακα.

ΒΙΒΛΙΟΓΡΑΦΙΑ – ΔΙΑΔΙΚΤΥΑΚΟΙ ΤΟΠΟΙ

[1] https://nodejs.org/en/

[2] https://expressjs.com/

[3] https://www.npmjs.com/package/ejs

[4] https://www.mongodb.com/

[5] https://mongoosejs.com/

[6] https://getbootstrap.com/

[7] https://www.npmjs.com/package/express

[8] https://www.npmjs.com/package/mongoose

[9] https://www.npmjs.com/package/ejs

[10] https://www.npmjs.com/package/joi

[11] https://www.npmjs.com/package/ejs-mate

[12] https://www.npmjs.com/package/express-session

[13] https://www.passportjs.org/

[14] https://www.npmjs.com/package/passport

[15] https://www.npmjs.com/package/passport-local

[16] https://www.npmjs.com/package/flash

[17] https://www.npmjs.com/package/multer

[18] https://cloudinary.com/

[19] https://www.npmjs.com/package/cloudinary

[20] https://www.npmjs.com/package/multer-storage-cloudinary

[21] https://www.mapbox.com/

[22] https://www.npmjs.com/package/@mapbox/mapbox-sdk

[23] https://github.com/LunarLogic/starability

[24] https://www.npmjs.com/package/express-mongo-sanitize

[25] https://www.npmjs.com/package/sanitize-html

[26] https://www.npmjs.com/package/helmet

[27] https://www.npmjs.com/package/dotenv

[28] https://www.npmjs.com/package/connect-mongo

[29] https://www.heroku.com/

[30] https://git-scm.com/

[31] https://en.wikipedia.org/wiki/History_of_the_Internet

[32] https://developer.mozilla.org/en-US/

[33] https://www.javatpoint.com/session-vs-cookies