ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Υλοποίηση διαδικτυακής εφαρμογής, με τεχνολογίες REST API, Laravel, και JavaScript

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Του

ΓΕΩΡΓΙΑΔΗ ΔΗΜΗΤΡΗ

(ΑΕΜ: 2753)

Επιβλέπων:

Δημήτριος Ι. Βέργαδος

Επίκουρος Καθηγητής

Καστοριά Ιανουάριος - 2023

Η παρούσα σελίδα σκοπίμως παραμένει λευκή

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Υλοποίηση διαδικτυακής εφαρμογής, με τεχνολογίες REST API, Laravel, και JavaScript

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Του

ΓΕΩΡΓΙΑΔΗ ΔΗΜΗΤΡΗ

(ΑΕΜ: 2753)

Επιβλέπων:

Δημήτριος Ι. Βέργαδος

Επίκουρος Καθηγητής

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 19 / 01 / 2023

Νίκος Δημόκας	Δημήτριος Ι. Βέργαδος	Σπυρίδων Νικολάου
Επίκουρος Καθηγητής	Επίκουρος Καθηγητής	Λέκτορας

Καστοριά Ιανουάριος - 2023

Ευχαριστίες

Κάπου εδώ τελείωσε μεγάλος αγώνας μου για την πτυχιακή εργασία. Θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στον καθηγητή και εισηγητή μου κ. Βέργαδο Δημήτριο, ο οποίος ήταν δίπλα μου, μου έλυνε απορίες, και με βοήθησε καθ όλην την διάρκεια της Πτυχιακής μου εργασίας.

Τέλος θα ήθελα να ευχαριστήσω θερμά την οικογένεια μου, που ήταν δίπλα μου καθημερινά και με στήριζαν ψυχολογικά και μου έδιναν δύναμη ώστε να δώσω το καλύτερο δυνατό αποτέλεσμα πάνω στην εργασία μου.

Περίληψη

Στην παρούσα πτυχιακή εργασία έχει μελετηθεί ένας από τους πολλούς τρόπους δημιουργίας μιας ολοκληρωμένης διαδικτυακής εφαρμογής καθώς και μερικές τεχνολογίες που μας κάνουν την ζωή μας πιο εύκολη και παράλληλα πιο ασφαλής όσον αφορά στην δημιουργία της εφαρμογής και στον τρόπο υλοποίησης της. Στην ανάπτυξη αυτής της εφαρμογής θα χρησιμοποιηθούν σύγχρονοι μέθοδοι υλοποίησης εφαρμογών και σύγχρονες τεχνολογίες και θα καλυφθεί ένα αρκετά καλό ποσοστό από τις δυνατότητες και τις λειτουργίες που προσφέρουν αυτές οι τεχνολογίες.

Λέξεις Κλειδιά: *Laravel, Vue, UI, JavaScript, PHP, HTML, CSS, MySQL, REST API, Framework* Apache, front-end, back-end, server.

Abstract

In this dissertation we have studied one of the many ways to create a complete web application as well as some technologies that make our lives easier and at the same time more secure. In the development of this application, modern methods of application implementation and modern technologies will be used and a fairly good percentage of the possibilities and functions offered by these technologies will be covered.

Key Words: *Laravel, Vue, UI, JavaScript, PHP, HTML, CSS, MySQL, REST API, Framework* Apache, front-end, back-end, server.

Πίνακας Περιεχομένων

Λίστα Εικόνων

Εισαγωγή

Η εργασία αυτή χωρίζεται και **5** κεφάλαια :

Στο **1ο** κεφάλαιο αναλύεται η γενική περιγραφή του συστήματος που έχει υλοποιηθεί και οι λόγοι της αναγκαιότητας / χρησιμότητας της εφαρμογής .

Το **2ο** κεφαλαίο αναλύει και εστιάζει στις τεχνολογίες που είναι απαραίτητες για την υλοποίηση της διαδικτυακής εφαρμογής .

Το **3ο** κεφάλαιο παρουσιάζει την σχεδίαση και την δομή του συστήματος με διαγράμματα και εικόνες .

Στο **4ο** κεφάλαιο παρουσιάζεται η υλοποίηση της διαδικτυακής εφαρμογής με παραδείγματα κώδικα και εικόνες με την λειτουργιά του .

Στο **5ο** και τελευταία κεφάλαιο της εργασίας καταγράφονται τα συμπεράσματα της εργασίας καθώς και διάφοροι πιθανοί στόχοι για το πως η εφαρμογή μπορεί να γίνει καλύτερη η με περισσότερες λειτουργίες .

1. Διαδικτυακή εφαρμογή

1.1 Ορισμός Της Διαδικτυακής εφαρμογής

Διαδικτυακή εφαρμογή [1] (web application) ονομάζεται κάθε εφαρμογή η ποία είναι διαθέσιμη στους χρήστες της μέσω του Διαδικτύου (Internet) και ο χρήστης χρειάζεται μόνο τον περιηγητή (browser) του για να την χρησιμοποιήσει . Οι εφαρμογές αυτές συνήθως εκτελούνται σε ισχυρές υπολογιστικές μηχανές οι οποίες έχουν τον ρόλο του σταθμού εξυπηρέτησης (server) και παρέχουν τις υπηρεσίες τους σε περισσότερους του ενός χρήστη .

1.2 Πλεονεκτήματα μιας Διαδικτυακής εφαρμογής

Με την πάροδο του χρόνου όλο και περισσότερες διαδικτυακές εφαρμογές κάνουν την εμφάνιση τους από την πιο απλή ηλεκτρονική ιστοσελίδα διαφήμισης κάποιου προϊόντος η κάποιας υπηρεσίας μέχρι την ποιο πολύπλοκη εφαρμογή που θα μπορούσε να είναι μια συνδρομητική υπηρεσία τύπου Netflix που έχει ενεργό κοινό από χρήστες που το χρησιμοποιούν καθημερινά διότι εξυπηρετεί την ανάγκη του κόσμου . Οπότε το συμπέρασμα είναι ότι υπάρχει η ανάγκη για μελέτη , κατανόηση και δημιουργία εφαρμογών στο διαδίκτυο. Μερικά από τα πλεονεκτήματα της δημιουργίας μιας διαδικτυακής εφαρμογής είναι :

1.2.1 Άμεση πρόσβαση ανεξάρτητα την συσκευή:

Οι χρήστες αυτών των εφαρμογών διαδικτύου έχουν άμεση και γρήγορη πρόσβαση στις εφαρμογές που χρησιμοποιούν από φορητό υπολογιστή ή γενικά μια συσκευή που διαθέτει ιντερνέτ. Η μόνη απαραίτητη εφαρμογή είναι ο περιηγητής ο οποίος είναι συνήθως προ εγκατεστημένος σε όλα τα λειτουργικά συστήματα ακόμα και στα κινητά τηλέφωνα. Η ιδιότητα αυτή των διαδικτυακών εφαρμογών είναι ιδιαίτερα σημαντική διότι στην περίπτωση της τοπικής εφαρμογής θα έπρεπε να εγκατασταθεί η εφαρμογή σε κάθε ένα υπολογιστή ξεχωριστά.

1.2.2 Δυνατότητα χρήσης από οποιαδήποτε τοποθεσία:

Οι χρήστες μπορούν να χρησιμοποιούν τις εφαρμογές σε οποιονδήποτε χώρο βρίσκονται αρκεί να υπάρχει σύνδεση στο διαδίκτυο . Η δυνατότητα αυτή δίνει ευελιξία χωρίς να δεσμεύεσαι να είσαι στο σπίτι η στο γραφείο.

1.2.3 Συμβατές με όλα τα λειτουργικά συστήματα:

Ένα ακόμα πλεονέκτημα των διαδικτυακών εφαρμογών είναι ότι είναι συμβατές με όλα τα λειτουργικά συστήματα. Καθώς η εφαρμογή εκτελείται μέσω του περιηγητή του

διαδικτύου και όχι στον υπολογιστή του χρήστη, την κάνει ικανή να εκτελείται σε όλα τα λειτουργικά συστήματα.

1.2.4 Δεν καταλαμβάνουν χώρο:

Οι διαδικτυακές εφαρμογές αυτές δεν καταλαμβάνουν καθόλου ή σχεδόν καθόλου χώρο στον δίσκο του χρήστη αφού το σύνολο της εφαρμογής είναι αποθηκευμένο στον εξυπηρετητή και μόνο κατά την χρήση της εφαρμογής μπορεί να υπάρχει μεταφορά δεδομένων προς την υπολογιστική μονάδα του χρήση και μόνο στην περίπτωση που ο χρήστης το επιθυμεί.

1.3 Εισαγωγή στο NoteBook Application

Αφού μιλήσαμε για τις διαδικτυακές εφαρμογές είναι η ώρα να σας κάνω μια εισαγωγή στην εφαρμογή που έχω υλοποίηση εγώ για αυτήν την πτυχιακή εργασία. Η εφαρμογή αυτή ονομάζεται NoteBook και είναι μια διαδικτυακή εφαρμογή στην οποία ο χρήστης πρώτα από όλα θα έχει την δυνατότητα να δημιουργήσει έναν λογαριασμό και μόλις το κάνει αυτό θα μπορεί να χρησιμοποιήσει την εφαρμογή και τις λειτουργίες της. Άρα δηλαδή υπάρχει ένα σύστημα αυθεντικοποίησης που θα ελέγχει αν ο χρήστης έχει δημιουργήσει λογαριασμό για να μπορέσει να χρησιμοποίηση την εφαρμογή . Σε περίπτωση που έχει δημιουργήσει λογαριασμό απλά θα πρέπει να συνδεθεί με αυτόν για να συνεχίσει , σε διαφορετική περίπτωση θα πρέπει να δημιουργήσει έναν για να μπορέσει να χρησιμοποιήσει την εφαρμογή .

1.4 Χρησιμότητα της εφαρμογής

Η εφαρμογή αυτό που κάνει είναι να δίνει στον χρήστη την δυνατότητα να μπορεί δημιουργεί σημειώσεις , ιστορίες , καταγραφές , ή οτιδήποτε άλλο θέλει ο χρήστης να αποθηκεύσει σε ψηφιακή μορφή αντί να το κάνει σε ένα χαρτί. Ουσιαστικά ο χρήστης μπορεί να δημιουργήσει κάτι σαν ένα ψηφιακό βιβλίο η σημειωματάριο στο οποίο θα κρατάει πληροφορίες που θέλει εκείνος όπως: κείμενα , εικόνες , βίντεο από το YouTube, δικές τους ιστορίες ή οτιδήποτε άλλο θέλει ο χρήστης να γράψει και να το έχει αποθηκευμένο στην εφαρμογή. Σε κάθε αρχείο που δημιουργεί ο χρήστης καταγράφετε η ώρα και η ημερομηνία για την καλύτερη εμπειρία του χρήστη με την εφαρμογή.

1.5 Πλεονεκτήματα χρήσης της εφαρμογής

Μερικά από τα πλεονεκτήματα της εφαρμογής είναι :

1.5.1 Εξοικονόμηση χώρου αποθήκευσης

Παρέχει στον χρήστη μια εφαρμογή που μπορεί να αποθηκεύσει τις πληροφορίες που θέλει χωρίς να χρειαστεί να κατεβάσει στο κινητό του η στον υπολογιστή του κάποια εφαρμογή που το κάνει αυτό και να χρειαστεί να δαπανήσει αποθηκευτικό χώρο από το κινητό η τον υπολογιστή του .

1.5.2 Δυνατότητα συγγραφής άρθρου

Στον χρήστη δίνεται η δυνατότητα συγγραφής άρθρου με κάποιες βασικές λειτουργίες όπως:

- η δυνατότητα για ποιο έντονα γράμματα γνωστό σε όλους σαν 'Bold'
- η δυνατότητα για υπογράμμιση λέξεων.
- η δυνατότητα για πλαγίαση γραμμάτων .
- Η δυνατότητα αλλαγής μεγέθους γραμμάτων .
- Δυνατότητα εισαγωγής πινάκα.
- Δυνατότητα εισαγωγής εικόνας.
- Δυνατότητα κατηγοριοποίησης σε λίστα (bulleted list) και (Numbered List).
- Δυνατότητα εισαγωγής αποστασιοποιήσεων .

1.5.3 Δυνατότητα αποθήκευσης βίντεο από το YouTube

Στον χρήστη δίνεται η δυνατότητα να αντιγράψει οποιονδήποτε σύνδεσμο από το YouTube και να το αποθηκεύσει στην εφαρμογή με σκοπό να το έχει κρατημένο σε περίπτωση που το ξεχάσει ή σε περίπτωση που γράφει κάποιο άρθρο και θέλει να το συμπεριλάβει στην δουλειά του ή για οποιοιδήποτε άλλο λόγο το θελήσει .

2. Εισαγωγή στις τεχνολογίες που χρησιμοποιηθήκαν

2.1 Διαχείριση δεδομένων

Σημασία της Διαχείρισης Δεδομένων [2] Τα δεδομένα στις μέρες μας καταγράφονται και χρησιμοποιούνται σε κάθε δραστηριότητα επιστημονική, εμπορική, παραγωγική, κυβερνητική, στρατιωτική κλπ. Οι μεγάλες ποσότητες δεδομένων επιβάλλουν την εύρεση αποτελεσματικών μεθόδων αποθήκευσης. Τα δεδομένα πρέπει να είναι οργανωμένα με τέτοιο τρόπο ώστε να διευκολύνεται η αναζήτηση και η ενημέρωσή τους. Ένας από τους βασικούς λόγους για τους οποίους αναθέτουμε σε υπολογιστές να επιλύουν προβλήματα είναι η δυνατότητα τους να αποθηκεύουν μεγάλο όγκο δεδομένων με ταχύτητα ακρίβεια και ασφάλεια. Εκτός από δυνατότητες εισαγωγής και αποθήκευσης δεδομένων στο σύστημα, δίνεται επιπλέον στον χρήστη η ευκαιρία να ανακαλεί, ενημερώνει, διαγράφει και γενικά να επεξεργάζεται και να αξιοποιεί με πολλούς τρόπους τα δεδομένα των αρχείων. Η διαχείριση των δεδομένων ανήκε στην ευθύνη του προγραμματιστή της εφαρμογής.

2.1.1 Εξέλιξη της Διαχείρισης Δεδομένων

Φυσική εξέλιξη της διαχείρισης δεδομένων [2] είναι τα Συστήματα Διαχείρισης Βάσεων Δεδομένων (ΣΔΒΔ) (Database Management Systems (DBMS)). Μια Βάση Δεδομένων ή ΒΔ θα μπορούσε να χαρακτηριστεί ως ένα σύνολο αρχείων τα οποία διαθέτουν υψηλό βαθμό οργάνωσης και είναι συνδεδεμένα μεταξύ τους με λογικές σχέσεις, ώστε να μπορούν να χρησιμοποιούνται από πολλές εφαρμογές και από πολλούς χρήστες. Τα αρχεία δε δημιουργούνται πλέον ούτε ενημερώνονται από ανεξάρτητες εφαρμογές λογισμικού αλλά από ένα ξεχωριστό σύστημα προγραμμάτων (λογισμικό). Το σύστημα αυτό μεσολαβεί ανάμεσα στα αρχεία δεδομένων και στις εφαρμογές που χρησιμοποιούν οι χρήστες και λέγεται Σύστημα Διαχείρισης Βάσης Δεδομένων ή ΣΔΒΔ (Data Base Management System DBMS). Το ΣΔΒΔ είναι ένα σύνολο προγραμμάτων και ρουτινών, που σκοπό έχουν το χειρισμό της βάσης, όσον αφορά τη δημιουργία, συντήρηση, επεξεργασία στοιχείων, ελέγχους ασφαλείας κτλ., και την εξυπηρέτηση των χρηστών, όσον αφορά την παροχή στοιχείων και πληροφοριών, χωρίς αυτοί να πρέπει να ασχολούνται με το πώς και το πού τα δεδομένα είναι αποθηκευμένα στη βάση.

Βάσεις δεδομένων που χρησιμοποιούνται :

- MySQL
- MSSQL
- MongoDB
- PostgresSQL

2.1.2 Εφαρμογές Βάσεων Δεδομένων στην Καθημερινή Ζωή

- Κρατήσεις θέσεων σε αεροπορικές εταιρείες.

Οι εφαρμογές αυτές υποστηρίζουν λειτουργίες [2], όπως κράτηση θέσης για μία συγκεκριμένη πτήση, αναζήτηση πληροφοριών για διαθέσιμες πτήσεις με βάση την αφετηρία και τον προορισμό, αναζήτηση πληροφοριών σχετικά με τις τιμές και τη διαθεσιμότητα των εισιτηρίων. Η λειτουργία της εφαρμογής επιτρέπει την απάντηση ερωτημάτων που αφορούν στην ώρα αναχώρησης και άφιξης συγκεκριμένων πτήσεων και την αποδοτική κράτηση θέσεων και έλεγχο της διαθεσιμότητας. Οι εφαρμογές αυτές είναι πολύτιμες για ταξιδιωτικούς πράκτορες και αεροπορικές εταιρείες λόγω των πολλών διευκολύνσεων που παρέχουν.

- Τραπεζικές Συναλλαγές.(Ibank)

Στις εφαρμογές αυτές [2] η πληροφορία αποτελείται από ονόματα πελατών, διευθύνσεις, αριθμούς τραπεζικών λογαριασμών, υπόλοιπο λογαριασμών, δεδομένα που αφορούν σε δάνεια, πιστωτικές κάρτες και γενικά από οτιδήποτε σχετίζεται με τη λειτουργία μίας τράπεζας. Στόχος των εφαρμογών αυτών είναι η ταχύτερη εξυπηρέτηση των πελατών και η αποδοτικότερη λειτουργία της τράπεζας. Αναπτύσσονται ταχύτατα με νέα προϊόντα που βασίζονται στο διαδίκτυο και καταργούν την φυσική παρουσία σε κατάστημα. Προφανώς μια τέτοιου επιπέδου εφαρμογή πρέπει να υποστηρίζει ταυτόχρονες προσπελάσεις στα δεδομένα από πολλούς χρήστες.

- Διαχείριση Εταιρικών Δεδομένων.

Η καλή οργάνωση των δεδομένων [2] μίας μεγάλης εταιρείας έχει ουσιαστική και δυναμική συμβολή στην αποδοτική λειτουργία της. Η μεθοδική οργάνωση των δεδομένων επιδρά σε όλα τα τμήματα της εταιρείας και αυτή η δομημένη οργάνωση της πληροφορίας επιτρέπει τη γρήγορη αναζήτηση στοιχείων και την άμεση ενημέρωση σε τυχόν αλλαγές στα δεδομένα.

2.2 Full Stack Web Application

Μια ολοκληρωμένη διαδικτυακή εφαρμογή χωρίζεται σε 2 μεγάλες κατηγορίες στο Back-end και στο Front-end .

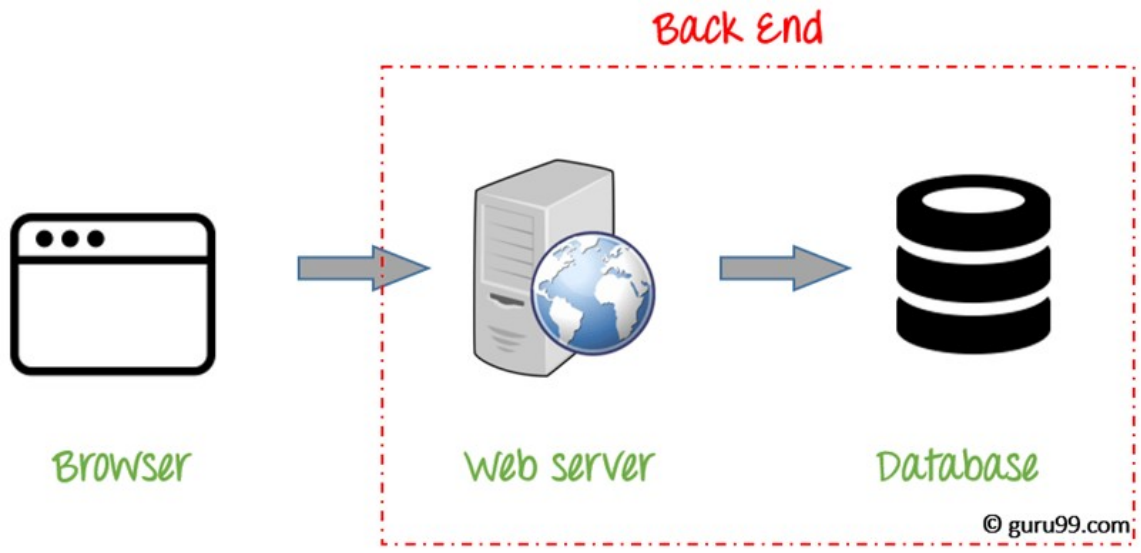
Το front-end ασχολείται κυρίως με το τμήμα UI (User Interface) του προγράμματος όπου οι διακομιστές (servers) αλληλεπιδρούν με τους χρήστες.

Το back-end είναι το μέρος του προγράμματος που συμβαίνει στην πλευρά του διακομιστή (Server). Το front-end μέρος είναι ορατό στο κοινό, ενώ το back-end δεν είναι. Ως εκ τούτου, είναι επίσης υπεύθυνο για τον τρόπο που λειτουργεί εσωτερικά.
Οι ιστότοποι που χρησιμοποιούμε στην καθημερινή μας ζωή είναι συνήθως ένας συνδυασμός γλωσσών όπως HTML, CSS και JavaScript που προστατεύονται από το πρόγραμμα περιήγησης που χρησιμοποιούμε και αποκρυπτογραφεί τον κώδικα έτσι ώστε να μπορεί να αλληλεπιδράσει εύκολα.

2.2.1 Back-end

Το Back-end Development [3] αναφέρεται στην πλευρά διακομιστή (Server). Επικεντρώνεται σε βάσεις δεδομένων και Περιέχει δραστηριότητες στο παρασκήνιο που εμφανίζονται κατά την εκτέλεση οποιασδήποτε ενέργειας σε έναν ιστότοπο. Μπορεί να είναι σύνδεση λογαριασμού ή πραγματοποίηση αγοράς από ηλεκτρονικό κατάστημα. Ο κώδικας που γράφτηκε από προγραμματιστές βοηθά τις διαδικτυακές εφαρμογές να επικοινωνούν με πληροφορίες βάσης δεδομένων.

Η τεχνολογία του back-end είναι ένας συνδυασμός διακομιστών, εφαρμογών και βάσεων δεδομένων.
Οι ευθύνες των προγραμματιστών back-end θα μπορούσαν να περιλαμβάνουν τη σύνταξη API, τη σύνταξη κώδικα για αλληλεπίδραση με μια βάση δεδομένων, τη δημιουργία βιβλιοθηκών, την εργασία σε επιχειρηματικές διαδικασίες και την αρχιτεκτονική δεδομένων και πολλά άλλα. Εξαρτάται συχνά από τον συγκεκριμένο ρόλο και την εταιρεία.



Εικόνα 1 Δομή Back-end

πηγή: https://www.guru99.com/what-is-backed-developer.html

2.2.2 REST API

Ας υποθέσουμε ότι προσπαθείτε να βρείτε βίντεο σχετικά με το ποδόσφαιρο στο Youtube. Ανοίγετε το Youtube, πληκτρολογείτε "ποδόσφαιρο" σε ένα πεδίο αναζήτησης, πατήστε enter και βλέπετε μια λίστα με βίντεο σχετικά με το ποδόσφαιρο.

Ένα REST API λειτουργεί [4] με παρόμοιο τρόπο. Αναζητάτε κάτι και λαμβάνετε μια λίστα αποτελεσμάτων από την υπηρεσία από την οποία ζητάτε. Το API είναι (application programming interface). Είναι ένα σύνολο κανόνων που επιτρέπουν στα προγράμματα να μιλούν μεταξύ τους. Ο προγραμματιστής δημιουργεί το API στο διακομιστή (Server) και επιτρέπει στον πελάτη να μιλήσει σε αυτόν.

Το REST καθορίζει [4] την εμφάνιση του API. Αντιπροσωπεύει την «Representational State Transfer» Είναι ένα σύνολο κανόνων που ακολουθούν οι προγραμματιστές όταν δημιουργούν το API τους. Ένας από αυτούς τους κανόνες δηλώνει ότι θα πρέπει να έχετε τη δυνατότητα να λάβετε ένα κομμάτι δεδομένων (resourse) όταν συνδέεστε σε μια συγκεκριμένη διεύθυνση URL. Κάθε διεύθυνση URL ονομάζεται αίτημα (request) ενώ τα δεδομένα που αποστέλλονται σε εσάς ονομάζονται απόκριση (response).

2.2.3 Requests

Ένα αίτημα αποτελείται από τέσσερα πράγματα

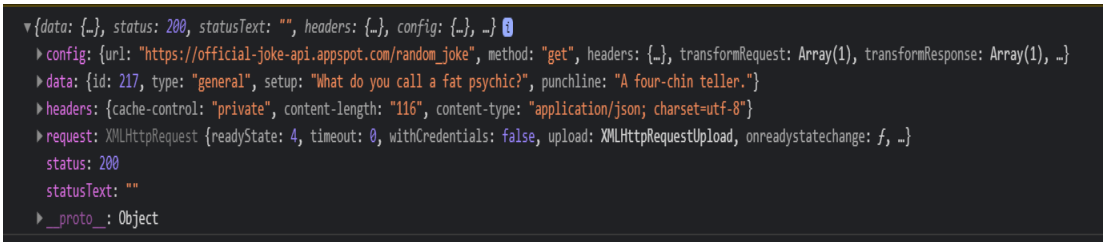
Το τελικό σημείο (endpoint)

Η μέθοδος (methods)

Οι κεφαλίδες (Headers)

Τα δεδομένα (data)

Το Status Code



Εικόνα 2 Απόκριση API

- **Endpoint**

  Το τελικό σημείο (endpoint)

  είναι το URL για το οποίο ζητάτε. πχ https://example/api/movies

- **Methods**

  Η μέθοδος (Method)

  είναι ο τύπος του αιτήματος που στέλνετε στο διακομιστή (Server).

  υπάρχουν 5 τύποι μεθόδων .

1. GET
2. POST
3. PUT
4. PATCH
5. DELETE

GET Μέθοδος

Αυτό το αίτημα χρησιμοποιείται για τη λήψη ενός η πολλών αποτελεσμάτων από έναν διακομιστή. Εάν εκτελέσετε ένα αίτημα «GET», ο διακομιστής αναζητά τα δεδομένα που ζητήσατε και τα στέλνει πίσω σε εσάς. Με άλλα λόγια, ένα αίτημα «GET» εκτελεί μια λειτουργία «ΑΝΑΓΝΩΣΗ».

POST Μέθοδος

Αυτό το αίτημα χρησιμοποιείται για τη δημιουργία ενός νέου δεδομένου σε έναν διακομιστή. Εάν εκτελέσετε ένα αίτημα «POST», ο διακομιστής δημιουργεί μια νέα καταχώριση στη βάση δεδομένων και σας λέει εάν η δημιουργία είναι επιτυχής. Με άλλα λόγια, ένα αίτημα «POST» εκτελεί μια λειτουργία «ΔΗΜΙΟΥΡΓΙΑ».

PUT & **PUTCH** Μέθοδος

Αυτά τα δύο αιτήματα χρησιμοποιούνται για την ενημέρωση ενός δεδομένου σε έναν διακομιστή. Εάν εκτελέσετε ένα αίτημα «PUT» ή «PATCH», ο διακομιστής ενημερώνει μια καταχώριση στη βάση δεδομένων και σας ενημερώνει εάν η ενημέρωση είναι επιτυχής. Με άλλα λόγια, ένα αίτημα «PUT» ή «PATCH» εκτελεί μια λειτουργία «ΕΝΗΜΕΡΩΣΗ». Η κύρια διαφορά τους είναι ότι το «PUT» αντικαταστεί ολόκληρη την πληροφορία ακόμα και αν θέλουμε να αλλάξουμε μόνο ένα συγκεκριμένο πεδίο. Ενώ το

«PATCH» αντικαταστεί μόνο το συγκεκριμένο πεδίο χωρίς να χρειάζεται να δώσουμε όλη την πληροφορία απλά για να αλλάξουμε ένα συγκεκριμένο κομμάτι.

DELETE μέθοδος

Αυτό το αίτημα χρησιμοποιείται για τη διαγραφή ενός δεδομένου από έναν διακομιστή. Εάν εκτελέσετε ένα αίτημα «ΔΙΑΓΡΑΦΗ», ο διακομιστής διαγράφει μια καταχώριση στη βάση δεδομένων και σας ενημερώνει εάν η διαγραφή είναι επιτυχής. Με άλλα λόγια, μια αίτηση «DELETE» εκτελεί μια λειτουργία «ΔΙΑΓΡΑΦΗ».

Headers

Οι κεφαλίδες χρησιμοποιούνται για την παροχή πληροφοριών τόσο στον χρήστη όσο και στον διακομιστή. Μπορεί να χρησιμοποιηθεί για πολλούς σκοπούς, όπως ο έλεγχος ταυτότητας και η παροχή πληροφοριών σχετικά με το περιεχόμενο του σώματος. Οι περισσότεροι διακομιστές πλέον έχουν περιεχόμενο τύπου JSON.

Data

Το data είναι το περιεχόμενο του API το οποίο καταλήγει να βλέπει ο χρήστης. Είναι αυτό που ουσιαστικά ζητάμε όταν στέλνουμε αίτημα τον διακομιστή οπότε και μας γυρνάει ένα αντίγραφο μέσα από την βάση.

2.2.4 HTTP Status Codes And Error Messages

Οι κώδικες κατάστασης HTTP [4] είναι κώδικες που εμφανίζονται μαζί με το εκάστοτε αίτημα και μας ενημερώνουν για το αποτέλεσμα που είχε το αίτημα μας . Έχουν την μορφή αριθμού και ο κάθε αριθμός σημαίνει κάτι διαφορετικό .

Οι ποιο συνηθισμένοι κωδικοί είναι :

- 200 σημαίνει ότι το αίτημα έχει πετύχει.
- 300 σημαίνει ότι το αίτημα ανακατευθύνεται σε άλλη διεύθυνση URL
- 400 σημαίνει ότι προέκυψε ένα σφάλμα που προέρχεται από τον πελάτη
- 500 σημαίνει ότι προέκυψε σφάλμα που προέρχεται από το διακομιστή

2.2.5 Back-end γλώσσες προγραμματισμού

Καθώς ο εγκέφαλός μας [5][6] μπορεί να κατανοήσει και να επεξεργαστεί ορισμένες γλώσσες που έχουμε μάθει με την πάροδο των ετών, το ίδιο ισχύει και για τον εγκέφαλο του υπολογιστή. Μπορεί να αναλύσει ορισμένες γλώσσες σε σύνολα εντολών και στη

συνέχεια να επεξεργαστεί αυτές τις εντολές.
Αυτές οι γλώσσες που μετατρέπονται σε σύνολα εντολών ονομάζονται γλώσσες προγραμματισμού. Αυτά χρησιμοποιούνται με συγκεκριμένους μεταγλωττιστές και σύνταξη για να επιτρέπεται στον υπολογιστή να τα επεξεργάζεται.

Οι γλώσσες προγραμματισμού που χρησιμοποιεί περισσότερο ο κόσμος για το back-end τμήμα του προγράμματος τους είναι :

- Java
- PHP
- Python
- C#
- JavaScript

Όλες αυτές οι γλώσσες προγραμματισμού κάνουν ακριβώς το ίδιο πράγμα όσον αφορά το back-end κομμάτι μιας διαδικτυακής εφαρμογής απλά την κάνουν με διαφορικό τρόπο και η κάθε μια με την δική της σύνταξη .

Οπότε μια από αυτές τις γλώσσες προγραμματισμού και μια βάση δεδομένων από αυτές που αναφέρθηκαν ποιο πάνω μας αρκεί για την δημιουργία του back-end κομματιού μιας διαδικτυακής εφαρμογής.

2.2.6 Front-End

Το front-end είναι το μπροστινό μέρος ενός ιστότοπου με το οποίο αλληλεπιδρούν οι χρήστες. Όλα όσα βλέπετε όταν περιηγείστε στο Διαδίκτυο, από γραμματοσειρές και χρώματα έως αναπτυσσόμενα μενού και sliders εικόνων, είναι ένας συνδυασμός HTML, CSS και JavaScript που ελέγχεται από το πρόγραμμα περιήγησης του υπολογιστή σας.

Οι προγραμματιστές front-end είναι υπεύθυνοι για τον κώδικα ενός ιστότοπου που βλέπει ο χρήστης. Για την επίτευξη αυτών των στόχων, οι προγραμματιστές front-end πρέπει να είναι εξοικειωμένοι με τρεις κύριες γλώσσες: HTML, CSS και προγραμματισμός Javascript. Εκτός από την ευχέρεια σε αυτές τις γλώσσες, οι προγραμματιστές front-end πρέπει να είναι εξοικειωμένοι με τεχνολογίες όπως το Bootstrap η React.js ή η Vue.js τα οποία διασφαλίζουν εξαιρετικό περιεχόμενο ανεξάρτητα από τη συσκευή και επιτρέπουν στις σελίδες να φορτώνουν δυναμικά περιεχόμενα κάνοντας λήψη δεδομένων από τον διακομιστή (Server).

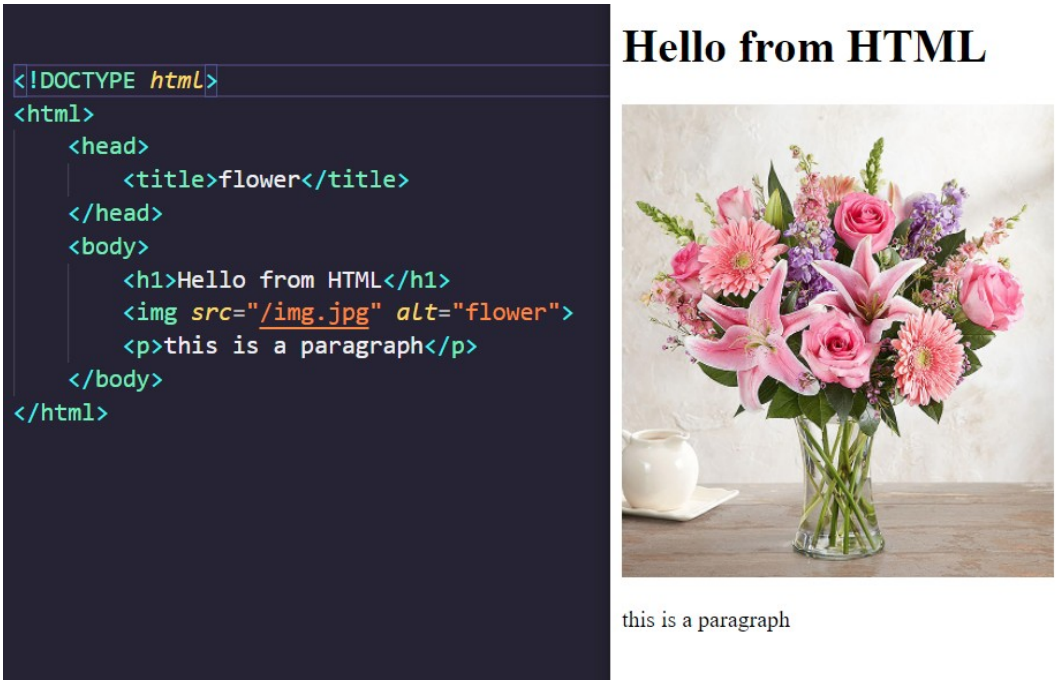
2.2.7 Εισαγωγή στην HTML

Το Hypertext Markup Language (HTML) είναι η γλώσσα που χρησιμοποιείται για έγγραφα που έχουν σχεδιαστεί για εμφάνιση σε πρόγραμμα περιήγησης στο Web. Μπορεί να υποβοηθηθεί από τεχνολογίες όπως Cascading Style Sheets (CSS) και

γλώσσες scripting όπως το JavaScript.Τα προγράμματα περιήγησης στο Web λαμβάνουν έγγραφα HTML από web server ή από τοπικό χώρο αποθήκευσης και αποδίδουν τα έγγραφα σε ιστοσελίδες πολυμέσων. Η HTML περιγράφει τη δομή μιας ιστοσελίδας σημασιολογικά και αρχικά περιελάμβανε στοιχεία για την εμφάνιση του εγγράφου.

2.2.8 Δομή της HTML

Η HTML παρέχει ένα μέσο για τη δημιουργία δομημένων εγγράφων υποδηλώνοντας τη δομική σημασιολογία για κείμενο όπως επικεφαλίδες, παραγράφους, λίστες, συνδέσμους, εισαγωγικά και άλλα στοιχεία. Τα στοιχεία HTML οριοθετούνται με ετικέτες, γραμμένες με αγκύλες. Ετικέτες όπως <img /> και <input /> εισάγουν απευθείας περιεχόμενο στη σελίδα. Άλλες ετικέτες όπως <p> και παρέχουν πληροφορίες σχετικά με το κείμενο του εγγράφου και μπορεί να περιλαμβάνουν άλλες ετικέτες ως υπο-στοιχεία. Τα προγράμματα περιήγησης δεν εμφανίζουν τις ετικέτες HTML, αλλά τις χρησιμοποιούν για να ερμηνεύσουν το περιεχόμενο της σελίδας.

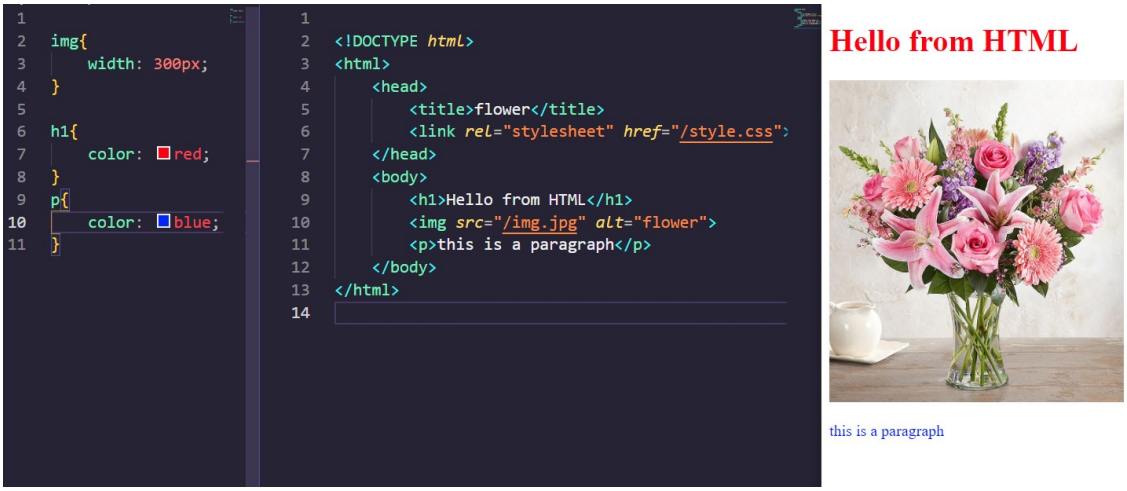


Εικόνα 3 HTML

2.2.9 Εισαγωγή στην CSS

Το Cascading Style Sheets (CSS) είναι μια γλώσσα που χρησιμοποιείται για την περιγραφή της παρουσίασης ενός εγγράφου γραμμένου σε γλώσσα όπως το HTML. Το CSS είναι ο ακρογωνιαίος λίθος του World Wide Web, παράλληλα με HTML και JavaScript. Το CSS έχει σχεδιαστεί για να επιτρέπει τον διαχωρισμό της παρουσίασης και

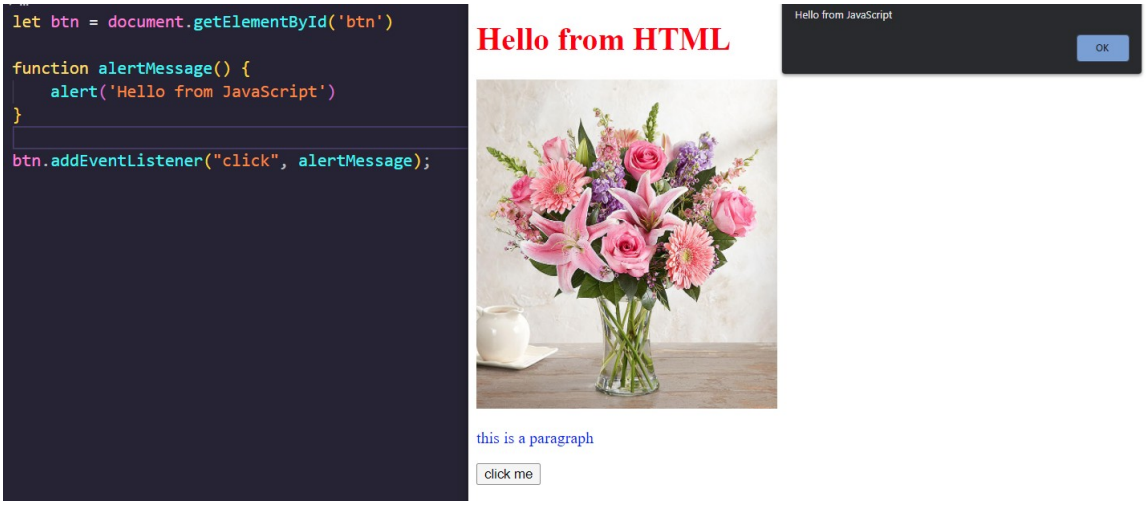
του περιεχομένου, συμπεριλαμβανομένης της διάταξης, των χρωμάτων και των γραμματοσειρών. Αυτός ο διαχωρισμός μπορεί να βελτιώσει την προσβασιμότητα του περιεχομένου, να παρέχει περισσότερη ευελιξία και έλεγχο στις προδιαγραφές των χαρακτηριστικών παρουσίασης, να επιτρέψει σε πολλές ιστοσελίδες να μοιράζονται τη μορφοποίηση καθορίζοντας το σχετικό CSS σε ένα ξεχωριστό αρχείο .css που μειώνει την πολυπλοκότητα και την επανάληψη στο δομικό περιεχόμενο καθώς και το αρχείο .css που θα αποθηκευτεί στην κρυφή μνήμη για να βελτιώσει την ταχύτητα φόρτωσης της σελίδας μεταξύ των σελίδων που μοιράζονται το αρχείο και τη μορφοποίησή του.



Εικόνα 4 CSS

2.2.10 Εισαγωγή στην JavaScript

Η JavaScript είναι μια γλώσσα προγραμματισμού που σας επιτρέπει να εφαρμόζετε σύνθετες λειτουργίες σε ιστοσελίδες κάθε φορά που μια ιστοσελίδα κάνει κάτι περισσότερο από το να εμφανίζει στατικές πληροφορίες για να δείτε εμφάνιση διαδραστικού περιεχομένου, διαδραστικών χαρτών, κινουμένων σχεδίων 2D / Τρισδιάστατα γραφικά, κύλιση βίντεο jukeboxes κ.λπ. - μπορείτε να στοιχηματίσετε ότι πιθανώς εμπλέκεται JavaScript. Είναι το τρίτο στρώμα του στρώματος των τυπικών τεχνολογιών Ιστού, δύο εκ των οποίων (HTML και CSS).



Εικόνα 5 JavaScript

2.2.11 Εισαγωγή στην PHP

Η PHP είναι μία γλώσσα σεναρίων, που τρέχει στον web server που βρίσκεται εγκατεστημένο το site. Χρησιμοποιείται για τον προγραμματισμό διαδικτυακών εφαρμογών και ιστοσελίδων με δυναμικό περιεχόμενο. Δεν διαμορφώνει αισθητικά τη σελίδα, αλλά παρέχει δυνατότητες όπως η εγγραφή χρηστών σε μία βάση δεδομένων και η εμφάνιση της τοπικής ώρας του server. Η εκτέλεση της PHP γίνεται με τη σειρά εμφάνισης του κώδικα, εκτός από τις συναρτήσεις που εκτελούνται όταν κληθούν. Ο κώδικας γράφεται μεταξύ των ετικετών αρχής "<?php>" και λήξης "?>". Για τις μεταβλητές δεν χρειάζεται να δηλωθεί ο τύπος τους και ξεκινούν με το σύμβολο για το δολάριο ($). Μεγάλη προσοχή δίνεται μεταξύ κεφαλαίων και πεζών, καθώς η PHP είναι case sensitive. Η PHP μπορεί να ενσωματωθεί στα HTML αρχεία, τα οποία αποθηκεύονται και καλούνται με την κατάληξη .php . Η ενσωμάτωση τους μπορεί να γίνει είτε σε οποιοδήποτε σημείο του HTML αρχείου, με τη δυνατότητα να εκτελείται υπό συγκεκριμένες συνθήκες, όπως για παράδειγμα το πάτημα ενός κουμπιού, είτε με την εντολή "include" με την οποία θα εκτελείται ένα εξωτερικό αρχείο PHP κάθε φορά που θα ανοίγει το HTML αρχείο. Όταν ζητηθούν τα αρχεία με κατάληξη .php από τον Client, ο web server καλεί τον PHP Interpreter και του παρέχει το .php αρχείο και πληροφορίες από τις κεφαλίδες τις HTTP αίτησης. Εκτελείται ο κώδικας PHP και το αποτέλεσμα του μπαίνει στον HTML κώδικα, ο οποίος μεταφέρεται στον buffer εξόδου. Ο τελευταίος μαζί με τις πληροφορίες κεφαλίδων της HTTP αίτησης, δίνονται στον Web Server και αυτός τις στέλνει στον Client με χρήση του πρωτοκόλλου HTTP. Αποτέλεσμα αυτών είναι η εμφάνιση στον browser μίας κοινής HTML σελίδας.

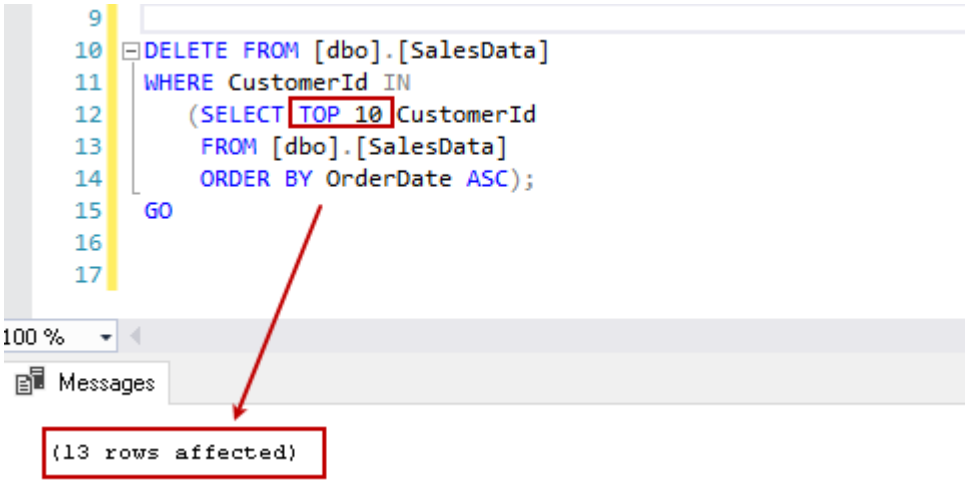
```php
public function index()
{
    try {
        $movies = Movie::all();

        return Response([
            'data' => $movies
        ], 200);
    } catch (Exception $error) {

        return Response([
            'message' => "No Movies Found !",
            'Error' => $error->getMessage()
        ], 404);
    }
}
```

Εικόνα 6 PHP

2.2.12 Εισαγωγή στην SQL

Η SQL σημαίνει δομημένη γλώσσα ερωτημάτων. Η SQL χρησιμοποιείται για την επικοινωνία με μια βάση δεδομένων. Σύμφωνα με το ANSI (Αμερικανικό Εθνικό Ινστιτούτο Προτύπων), είναι η τυπική γλώσσα για συστήματα διαχείρισης σχεσιακών βάσεων δεδομένων. Οι δηλώσεις SQL χρησιμοποιούνται για την εκτέλεση εργασιών όπως η ενημέρωση δεδομένων σε μια βάση δεδομένων ή η ανάκτηση δεδομένων από μια βάση δεδομένων. Μερικά κοινά συστήματα διαχείρισης σχεσιακών βάσεων δεδομένων που χρησιμοποιούν SQL είναι: Oracle (MySQL), Microsoft SQL Server, PostgreSQL .

Αν και τα περισσότερα συστήματα βάσεων δεδομένων χρησιμοποιούν SQL, τα περισσότερα από αυτά έχουν επίσης τις δικές τους πρόσθετες ιδιόκτητες επεκτάσεις που συνήθως χρησιμοποιούνται μόνο στο σύστημά τους. Ωστόσο, οι τυπικές εντολές SQL όπως "Select", "Insert", "Update", "Delete", "Create" και "Drop" μπορούν να χρησιμοποιηθούν για να ολοκληρώσουν σχεδόν όλα όσα χρειάζεται να κάνει κάποιος με μια βάση δεδομένων.



Εικόνα 7 SQL

πηγη : https://www.google.com/search?q=sql+code&source=lnms&tbm=isch&sa=X&ved=2ahUKEwiF4deE7cP2AhU2SfEDHV2PAC8Q_AUoAXoECAEQAw&biw=1309&bih=636&dpr=1.1#imgrc=TNcJrHcHw0vwNM

2.2.13 Εισαγωγή στην Node.js

Για τα πρώτα 20 χρόνια, η JavaScript χρησιμοποιήθηκε κυρίως για την εκτέλεση κώδικα σε διαδικτυακό περιβάλλον περιβάλλον φυλλομετρητή (web browser) Δεδομένου ότι η JavaScript μπορούσε να χρησιμοποιηθεί μόνο εντός της HTML ετικέτας <script>, οι προγραμματιστές έπρεπε να εργάζονται σε πολλαπλές γλώσσες για να δημιουργούσουν μια ολοκληρωμένη full stack εφαρμογή . Αργότερα ήρθε το Node.js, το οποίο είναι ένα περιβάλλον εκτέλεσης κώδικα JavaScript για την πλευρά του server όπως άλλες γλώσσες προγραμματισμού πχ Java, PHP, καθώς παρέχει όλα τα απαραίτητα εργαλεία για ώστε κάτι τέτοιο να γίνει εφικτό. Το Node.js είναι ένα ανοικτού κώδικα περιβάλλον εκτέλεσης με ένα μόνο νήμα (thread), ανοικτού κώδικα και πολλαπλών πλατφορμών για τη δημιουργία γρήγορων και κλιμακούμενων εφαρμογών στην πλευρά του διακομιστή (web server) και εφαρμογών δικτύωσης. Τρέχει με τη μηχανή εκτέλεσης V8 JavaScript και χρησιμοποιεί αρχιτεκτονική εισόδου/εξόδου που καθοδηγείται από συμβάντα και δεν μπλοκάρει, γεγονός που την καθιστά αποδοτική και κατάλληλη για εφαρμογές πραγματικού χρόνου.

2.2.14 Αρχιτεκτονική Node.js

Σε ένα συνδυασμό αίτησης-απόκρισης (request - response) με πολλαπλά νήματα (threads), πολλοί πελάτες στέλνουν μια αίτηση και ο διακομιστής (server) επεξεργάζεται κάθε μία πριν στείλει πίσω την απάντηση. Ωστόσο, χρησιμοποιούνται πολλαπλά νήματα (threads) για την επεξεργασία ταυτόχρονων κλήσεων. Ας ρίξουμε μια ματιά σε κάθε βήμα που περνάει:

- Το Node.js διατηρεί μια περιορισμένη δεξαμενή νημάτων για την εξυπηρέτηση των αιτήσεων.

- Κάθε φορά που έρχεται ένα αίτημα, το Node.js το τοποθετεί σε μια ουρά.
- Τώρα, το single-threaded "Event loop" το βασικό συστατικό περιμένει αιτήσεις επ' αόριστον.
- Όταν έρχεται ένα αίτημα, ο βρόχος το παίρνει από την ουρά και ελέγχει αν απαιτεί μια μπλοκαρισμένη λειτουργία εισόδου/εξόδου (I/O). Εάν όχι, επεξεργάζεται το αίτημα και στέλνει μια απάντηση.
- Ο βρόχος συμβάντος παρακολουθεί τις αιτήσεις που μπλοκάρουν και τις τοποθετεί στην ουρά μόλις η εργασία μπλοκαρίσματος επεξεργαστεί. Με αυτόν τον τρόπο διατηρεί τη μη φραγμένη φύση του.

Δεδομένου ότι το Node.js χρησιμοποιεί λιγότερα νήματα, χρησιμοποιεί λιγότερους πόρους/μνήμη, με αποτέλεσμα την ταχύτερη εκτέλεση εργασιών.

2.2.15 Χαρακτηριστικά του Node.js

Το Node.js έχει αναπτυχθεί γρήγορα τα τελευταία χρόνια. Αυτό οφείλεται στον τεράστιο κατάλογο των δυνατοτήτων που παρέχει:

Επεκτασιμότητα**:** Παρέχει τεράστια επεκτασιμότητα για τις εφαρμογές. Το Node.js, όντας single-threaded, είναι ικανό να χειρίζεται τεράστιο αριθμό ταυτόχρονων συνδέσεων με υψηλή απόδοση.

Ταχύτητα**:** Η εκτέλεση νημάτων χωρίς μπλοκαρίσματα καθιστά το Node.js ακόμη πιο γρήγορο και αποδοτικό.

Βιβλιοθήκες**:** Ένα τεράστιο σύνολο από πακέτα Node.js ανοικτού κώδικα είναι διαθέσιμα που μπορούν να απλοποιήσουν την εργασία σας. Υπάρχουν περισσότερα από ένα εκατομμύριο πακέτα στο οικοσύστημα NPM σήμερα.

Συντηρησιμότητα**:** Η Node.js είναι μια εύκολη επιλογή για τους προγραμματιστές, καθώς τόσο το frontend όσο και το backend μπορούν να διαχειριστούν με τη JavaScript ως ενιαία γλώσσα.

2.2.16 Μέγεθος αγοράς του Node.js

Τις τελευταίες 2 δεκαετίες έχει σημειωθεί τεράστια ανάπτυξη των ιστότοπων και, όπως ήταν αναμενόμενο, το Node.js αναπτύσσεται επίσης γρήγορα. Το δημοφιλές runtime ξεπέρασε ήδη το όριο του 1 δισεκατομμυρίου λήψεων το 2018, και σύμφωνα με το W3Techs, το Node.js χρησιμοποιείται από το 1,2% όλων των ιστότοπων παντού. Αυτό σημαίνει πάνω από 20 εκατομμύρια συνολικές τοποθεσίες σε όλο το διαδίκτυο.

Δεν αποτελεί έκπληξη το γεγονός ότι είναι μια δημοφιλής επιλογή και για εκατομμύρια εταιρείες. Ακολουθούν μερικές δημοφιλείς από αυτές που χρησιμοποιούν το Node.js σήμερα:

- Twitter

- Spotify
- eBay
- Reddit
- Netflix

2.2.17 Δυνατότητες του Node.js

Το Node.js χρησιμοποιείται για μια μεγάλη ποικιλία εφαρμογών. Ας εξερευνήσουμε μερικές δημοφιλείς περιπτώσεις χρήσης όπου το Node.js είναι μια καλή επιλογή:

Συζητήσεις σε πραγματικό χρόνο**:** Λόγω της ασύγχρονης φύσης του, το Node.js είναι κατάλληλο για την επεξεργασία επικοινωνίας σε πραγματικό χρόνο. Μπορεί εύκολα να κλιμακωθεί και χρησιμοποιείται συχνά στην κατασκευή chatbots. Το Node.js καθιστά επίσης απλή την κατασκευή πρόσθετων λειτουργιών συνομιλίας, όπως η συνομιλία πολλών ατόμων και οι ειδοποιήσεις push.

Internet of Things-IoT**:** περιλαμβάνουν συνήθως πολλούς αισθητήρες, καθώς συχνά στέλνουν μικρά κομμάτια δεδομένων που μπορούν να συσσωρευτούν σε μεγάλο αριθμό αιτημάτων. Το Node.js είναι μια καλή επιλογή, καθώς είναι σε θέση να χειριστεί γρήγορα αυτά τα ταυτόχρονα αιτήματα.

Ροή δεδομένων: Εταιρείες όπως το Netflix χρησιμοποιούν το Node.js για σκοπούς ροής δεδομένων. Αυτό οφείλεται κυρίως στο γεγονός ότι το Node.js είναι ελαφρύ και γρήγορο. Αυτές οι ροές επιτρέπουν στους χρήστες να διοχετεύουν αιτήματα μεταξύ τους, με αποτέλεσμα τα δεδομένα να διοχετεύονται απευθείας στον τελικό προορισμό τους.

Πολύπλοκες εφαρμογές μίας σελίδας (SPAs):-Στις SPAs, ολόκληρη η εφαρμογή φορτώνεται σε μία μόνο σελίδα. Αυτό συνήθως σημαίνει ότι υπάρχουν μερικές αιτήσεις που γίνονται στο παρασκήνιο για συγκεκριμένα στοιχεία. Το Node.js επεξεργάζεται τα αιτήματα με τρόπο που δεν μπλοκάρει.

Εφαρμογές βασισμένες σε REST API: Η JavaScript χρησιμοποιείται τόσο στο frontend όσο και στο backend των ιστότοπων. Έτσι, ένας διακομιστής μπορεί εύκολα να επικοινωνεί με το frontend μέσω REST APIs χρησιμοποιώντας το Node.js. Το Node.js παρέχει επίσης πακέτα όπως το Express.js και το Koa που διευκολύνουν ακόμη περισσότερο τη δημιουργία εφαρμογών ιστού.

2.2.18 NPM in Node.js

Το NPM είναι το οικοσύστημα πακέτων του Node.js. Είναι το μεγαλύτερο οικοσύστημα όλων των βιβλιοθηκών ανοιχτού κώδικα στον κόσμο, με πάνω από 1 εκατομμύριο πακέτα και αυξάνεται συνεχώς. Η χρήση του NPM είναι δωρεάν και χιλιάδες προγραμματιστές ανοιχτού κώδικα συνεισφέρουν καθημερινά σε αυτό. Το NPM έρχεται με ένα βοηθητικό πρόγραμμα γραμμής εντολών out-of-box. Μπορείτε απλά να μεταβείτε στον ιστότοπο της NPM για να αναζητήσετε το πακέτο που χρειάζεστε και να το

εγκαταστήσετε χρησιμοποιώντας μια μόνο εντολή. Μπορείτε επίσης να διαχειρίζεστε τις εκδόσεις του πακέτου σας μέσω της γραμμής εντολών. Χωρίς αμφιβολία, το NPM είναι το πιο αγαπημένο απόκτημα της κοινότητας του Node.js. Το Node.js προσελκύει μεγάλο αριθμό προγραμματιστών κυρίως λόγω της εξαιρετικής υποστήριξης πακέτων.

2.2.19 Εγκατάσταση πακέτων με το NPM μέσω CLI στο Node.js

Όταν εγκαθιστάτε το Node.js, το NPM εγκαθίσταται αυτόματα μαζί του. Εντολή για την εγκατάσταση ενός πακέτου με το NPM:

npm install <όνομα πακέτου>

Μπορείτε ακόμη και να εγκαταστήσετε πολλά πακέτα ταυτόχρονα:

npm install <pkg-1> <pkg-2> <pkg-3>

Μπορείτε επίσης να καθορίσετε τη σημαία -g (global) αν θέλετε να εγκαταστήσετε ένα πακέτο σε παγκόσμιο πλαίσιο. Αυτό σας επιτρέπει να χρησιμοποιήσετε το πακέτο οπουδήποτε στο μηχάνημά σας.

npm install -g <όνομα πακέτου>

Όταν αρχικοποιείτε μια νέα εφαρμογή, το NPM δημιουργεί αυτόματα ένα αρχείο package.json που αποτελείται από όλα τα πακέτα NPM. Εδώ είναι που μπορείτε να καθορίσετε εκδόσεις, εξαρτήσεις και προσαρμοσμένα σενάρια.

Υπάρχει ένας μακρύς κατάλογος εντολών που συνοδεύουν το βοηθητικό πρόγραμμα NPM, όπως η δημοσίευση, ο έλεγχος, η εκτέλεση και άλλα. Μπορείτε να ελέγξετε τον τρόπο χρήσης τους χρησιμοποιώντας την εντολή npm help.

2.2.20 Συμπεράσματα για το Node.js

Με λίγα λόγια, το Node.js είναι ένα δημοφιλές προγραμματιστικό περιβάλλον που μπορεί να χρησιμοποιηθεί για την κατασκευή εφαρμογών υψηλής κλίμακας που πρέπει να υποστηρίζουν πολλαπλά ταυτόχρονα αιτήματα. Το single-threaded non-blocking I/O το καθιστά επίσης μια εξαιρετική επιλογή τόσο για εφαρμογές πραγματικού χρόνου όσο και για εφαρμογές ροής δεδομένων. Για να το ενισχύσει, ακόμη περισσότερο, το Node.js διαθέτει μια τεράστια κοινότητα ενεργών προγραμματιστών και μπορεί να υπερηφανεύεται για το μεγαλύτερο αποθετήριο πακέτων ανοικτού κώδικα στον κόσμο, το NPM, το οποίο περιέχει σήμερα πάνω από ένα εκατομμύριο πακέτα.

2.3 Εξήγηση στην τεχνολογία SPA (Sinlge Page Application)

Με δεδομένο ότι πλέον υπάρχει μια αρχική κατανόηση πάνω σε HTML , CSS , JavaScript και γνώση για το τι είναι και τι κάνει το καθένα. Παρακάτω θα εξηγηθούν οι

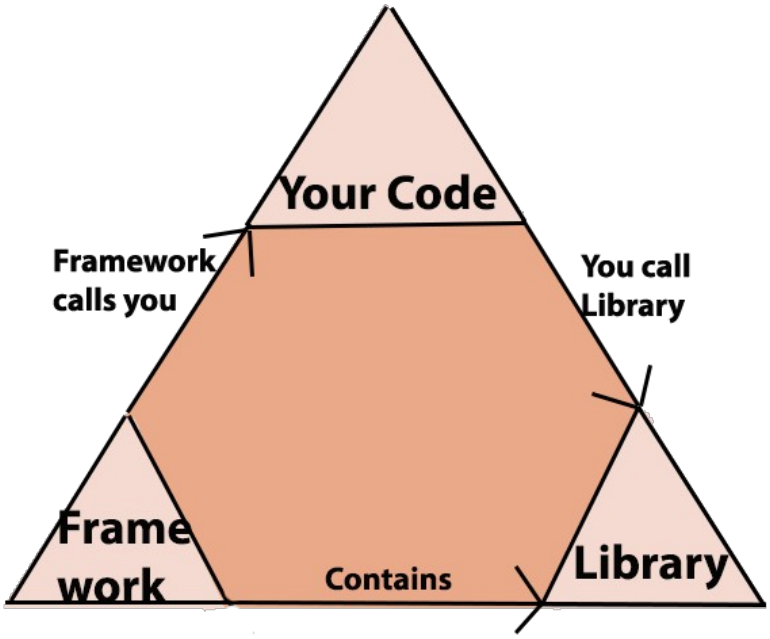
τεχνολογίες με τις οποίες έχει φτιαχτεί αυτή η διαδικτυακή εφαρμογή. Θα εξηγηθούν τι είναι που χρησιμεύουν και γιατί τα χρησιμοποίησα.

Η πλειονότητα των σύγχρονων ιστοσελίδων χρησιμοποιούν JavaScript σε κάποιο σχήμα ή μορφή, για να κάνουν τις σελίδες περισσότερες διαδραστικές και με περισσότερη λειτουργικότητα. Οι παραδοσιακές ιστοσελίδες είναι εφαρμογές πολλαπλών σελίδων, όπου ένα νέο έγγραφο HTML φορτώνεται κάθε φορά που ο χρήστης αλλάζει σελίδα ή το περιεχόμενό του. Αυτή είναι μια σχετικά αργή επιλογή σε σύγκριση με την πιο μοντέρνα έκδοση μοντέλο ανάπτυξης SPA (Single Page Application), όπου μόνο τα μέρη που αλλάζουν στο εκάστοτε αίτημα του χρήστη λαμβάνονται από το διακομιστή (Server) και ενημερώνονται κατάλληλα .

Η χρήση του μοντέλου SPA μειώνει τις ταχύτητες φόρτωσης εφαρμογών και βελτιώνει την εμπειρία του χρήστη καθώς η σελίδα φορτώνει τα νέα δεδομένα από τον Διακομιστή (Server) μόνο στο σημείο της σελίδας που έγινε η αλλαγή και όχι ανανέωση ολόκληρης της σελίδας όπως γινόταν παλιότερα. Αυτό έχει ως αποτέλεσμα η περιήγηση στη σελίδα να γίνεται πολύ γρηγορότερα.

2.3.1 Εισαγωγή στην έννοια Framework

Το Framework είναι μια πλατφόρμα που χρησιμοποιείται ως βάση για την ανάπτυξη εφαρμογών. Παρέχει μια δομή στην οποία οι προγραμματιστές λογισμικού μπορούν να δημιουργήσουν προγράμματα για μια συγκεκριμένη πλατφόρμα. Για παράδειγμα, ένα Framework μπορεί να περιλαμβάνει προκαθορισμένες κλάσεις και λειτουργίες που μπορούν να χρησιμοποιηθούν για την επεξεργασία, τη διαχείριση συσκευών υλικού και την αλληλεπίδραση με λογισμικό συστήματος. Αυτό βελτιστοποιεί τη διαδικασία ανάπτυξης, καθώς οι προγραμματιστές δεν χρειάζεται να ανακαλύπτουν ξανά τον τροχό κάθε φορά που αναπτύσσουν μια νέα εφαρμογή.



Εικονα 8 Παράδειγμα τριάδας You-Framework-Library

πηγη: https://benyoss4.medium.com/the-importance-of-frameworks-2c4a04d20ac5

2.3.2 Γιατί να χρησιμοποιήσω Framework

Η ανάπτυξη λογισμικού είναι μια πολύπλοκη διαδικασία. Απαιτεί πληθώρα εργασιών, όπως η συγγραφή του κώδικα , η σχεδίαση και η δοκιμή. Μόνο για το κομμάτι του κώδικα οι προγραμματιστές έπρεπε να φροντίσουν τη σύνταξη, τις δηλώσεις, τη συλλογή απορριμμάτων, τις εξαιρέσεις και άλλα.
Τα Frameworks διευκολύνουν τη ζωή των προγραμματιστών, επιτρέποντάς τους να αναλάβουν τον έλεγχο ολόκληρης της διαδικασίας ανάπτυξης λογισμικού, ή το μεγαλύτερο μέρος της, από μία μόνο πλατφόρμα. Για να συντομεύσουμε τα πράγματα, απαιτούνται frameworks όταν ο κώδικας για μια εφαρμογή φαίνεται να έχει μηδενική δομή και λειτουργικότητα, γιατί τα framewokrs, επιβάλλουν τον τρόπο δομής του κώδικα και πώς θα εκτελεστεί. Έχουν μια συγκεκριμένη δομή οργάνωση με φακέλους που περιέχουν μέσα τους όλα τα απαρέτητα αρχεία, αλλά και φυσικά χώρο για να δημιουργήσουμε τα δικά μας. Επίσης η χρήση των frameworks μας δίνει την δυνατότητα καλύτερης και πιο γρήγορης συντήρησης του προγραμμάτος καθώς υπάρχει οργάνωση και δομή στον τρόπο που είναι γραμμένος ο κώδικας.

Πλεονεκτήματα της χρήσης ενός Framework:

1. Βοηθά στην καθιέρωση καλύτερων πρακτικών προγραμματισμού και στην εφαρμογή των σχεδιαστικών προτύπων.
2. Ο κωδικός είναι πιο ασφαλής.
3. Διπλότυπος και περιττός κωδικός μπορεί να αποφευχθεί.
4. Βοηθά στη συνεπή ανάπτυξη κώδικα με λιγότερα σφάλματα.
5. Χρήση / διόρθωση του κώδικα και από τρίτους οι οποίοι δεν είχαν ανάμιξη στην αρχική εφαρμογή
6. Αρκετά τμήματα κώδικα και λειτουργίες είναι προ-δομημένα και προ-δοκιμασμένα. Αυτό καθιστά τις εφαρμογές πιο αξιόπιστες
7. Ο έλεγχος και ο εντοπισμός σφαλμάτων του κώδικα είναι πολύ ευκολότερος και μπορεί να γίνει ακόμη και από προγραμματιστές που δεν κατέχουν τον κώδικα
8. Ο χρόνος που απαιτείται για την ανάπτυξη μιας εφαρμογής μειώνεται σημαντικά

2.3.3 Εισαγωγή στο Framework VueJS

Το Vue.js είναι [8,9] μια βιβλιοθήκη της JavaScript, που δημιουργήθηκε από τον Evan You. Πριν δημιουργήσει το Vue, ο Evan εργαζόταν στην Google. Ενώ βρισκόταν στην Google, ο Evan θεώρησε ότι χρησιμοποιώντας συγκεκριμένα το Angular (ένα ακόμα δημοφιλής JavaScript framework) αποφάσισε να εξαγάγει τα μέρη που του άρεσαν από την Angular, για να δημιουργήσει την δική του βιβλιοθήκη . Το 2014 μοιράστηκε το έργο του με άλλους στο GitHub και έτσι Το Vue.js ξεκίνησε.

Για να ξεκινήσει κάποιος να μαθαίνει την Vue πρέπει να γνωρίζει τα βασικά: HTML, JavaScript και CSS. Μερικά χρόνια μετά το Vue.js είναι ένα από τα πιο δημοφιλή και ενεργά υποστηριζόμενα Framework με μεγάλο Community να την χρησιμοποιεί και να την στηρίζει .

2.3.4 Πλεονεκτήματα της Vue.js

- Μικρό μέγεθος
  επηρεάζει ς θετικά το SEO (search engine optimization) και το UX (User Experience) σας.

- Αμφίδρομη σύνδεση δεδομένων

  Με τη βοήθεια της αμφίδρομης δέσμευσης δεδομένων, είναι πιο εύκολο να ενημερώνονται στοιχεία και δεδομένα .

- όλα τα κομμάτια σε ένα αρχείο

Κάθε κομμάτι της εφαρμογής / ιστοσελίδας στο Vue είναι ένα στοιχείο. Στο Vue.js, τα στοιχεία μπορούν να γραφτούν σε HTML, CSS και JavaScript χωρίς να τα χωρίσουν σε ξεχωριστά αρχεία.

➔ Αυτό είναι καλό επειδή προσφέρει:

➔ Επαναχρησιμοποίηση κώδικα.

➔ Καλύτερη Αναγνωσιμότητα κώδικα.

➔ βοηθητικό σε περιβάλλον testing.

➔ Εύκολο στην εκμάθηση.

2.3.5 Εγκατάσταση και δομή της Vue

Για να εγκαταστήσει κάποιος την Vue στον υπολογιστή του πρώτα θα πρέπει να έχει εγκαταστήσει την Node.js η οποία περιέχει το npm (Node package manager) το οποίο είναι ο προεπιλεγμένος διαχειριστής πακέτων για το Node.js. Στην συνέχεια θα πρέπει να ανοίξει το terminal και να πληκτρολογήσει την εντολή vue create app, το <<vue create >> είναι η εντολή και το <<app>> είναι το όνομα του φακέλου όπου μέσα σε αυτόν θα δημιουργηθεί το vue application

Παρακάτω βλέπουμε ένα μικρο παράδειγμα χρησιμοποίησης της vue.

Εικόνα 9 Vue Component

2.3.6 Κατανόηση των υπολογισμένων ιδιοτήτων στο Vue.js

Στο Vue.js, υπάρχουν διάφοροι τρόποι για να ορίσουμε στατικές ή δυναμικές τιμές που θα εμφανίζονται στο template. Εδώ είναι που οι υπολογισμένες ιδιότητες έρχονται εξαιρετικά χρήσιμες. Σε αυτό το σεμινάριο, θα εξετάσουμε τα βασικά στοιχεία της χρήσης υπολογισμένων ιδιοτήτων στο Vue.js.

Μπορούμε να χρησιμοποιήσουμε υπολογιστικές ιδιότητες για να υπολογίσουμ και να εμφανίσουμε τιμές με βάση μια τιμή ή ένα σύνολο τιμών στο μοντέλο δεδομένων. Μπορείτε να χρησιμοποιήσετε υπολογισμένες ιδιότητες για να λύσετε πιο σύνθετα προβλήματα.

Φιλτράρισμα δεδομένων:

Οι υπολογισμένες ιδιότητες είναι εξαιρετικές για το φιλτράρισμα δεδομένων. Για παράδειγμα, ας πούμε ότι φιλτράρετε έναν πίνακα από μια γραμμή αναζήτησης εισόδου. Στην παρακάτω συνάρτηση δεδομένων, έχουμε έναν πίνακα, userData, που περιέχει πληροφορίες που θέλουμε να εμφανίσουμε στο στοιχείο, αλλά πρέπει επίσης να επιτρέψουμε στον χρήστη να φιλτράρει αυτό που εμφανίζεται κάνοντας αναζήτηση με μια ετικέτα εισόδου (δεσμευμένη με την ιδιότητα δεδομένων searchQuery). Όλα αυτά γίνονται μέσα στην υπολογισμένη ιδιότητα resultQuery:

computed: {

resultQuery() {

```
    if (this.searchQuery) {
      return this.userData.filter((item) => {
        return this.searchQuery
          .toLowerCase()
          .split(" ")
          .every((v) => item.name.toLowerCase().includes(v)),
      });
    } else {
      console.log(this.userData),
      return this.userData,
    }
  },
 },
};
```

2.3.7 Κατανόηση των γεγονότων στο Vue.js

Υπάρχουν αρκετοί διαφορετικοί τρόποι χειρισμού συμβάντων στη Vue, αλλά η καλύτερη λύση θα εξαρτηθεί από τον τύπο του συμβάντος που ακούμε, τον τρόπο με τον οποίο θέλουμε να αντιδράσουμε στο συμβάν και το τι ελπίζουε να επιτύχουμε με το συμβάν. Για παράδειγμα, αν ένας χρήστης κάνει κλικ σε ένα κουμπί, υποβιβάζει μια φόρμα ή ακόμα και αν απλά μετακινεί το ποντίκι του, μπορούμε να προσθέσουμε μια αντίδραση, όπως η προβολή ενός animation ή η κλήση μιας συνάρτησης. Μερικοί τροποποιητές συμβάντων ειναι :

Ο τροποποιητής .prevent εμποδίζει το άνοιγμα του συνδέσμου και εκτελεί μόνο τη μέθοδο που έχει εκχωρηθεί.

.self : ενεργοποιεί το συμβάν μόνο αν το event.target είναι το ίδιο το στοιχείο.

.once : αποτρέπει την εκτέλεση του συμβάντος περισσότερες από μία φορές.

.keyup : ακούει συμβάντα πληκτρολογίου.

.capture : χειρίζεται συμβάντα που στοχεύουν ένα εσωτερικό στοιχείο πριν από το χειρισμό του στοιχείου που προκάλεσε το συμβάν.

Αν και ο χειρισμός συμβάντων μπορεί να φαίνεται απλός, ένας λανθασμένος ή ελλιπής χειριστής συμβάντων μπορεί να προκαλέσει ακαταστασία στον κώδικά.

2.4 Εισαγωγή στην Laravel

Το Laravel είναι ένα web framework [10] που προσπαθεί να διευκολύνει τη διαδικασία ανάπτυξης μιας εφαρμογής και απλοποιεί σε μεγάλο βαθμό τις επαναλαμβανόμενες εργασίες που χρησιμοποιούνται στις περισσότερες από τις σημερινές εφαρμογές ιστού (web) συμπεριλαμβανομένων και προβλήματα όπως δρομολόγηση (routing), έλεγχο ταυτότητας (authentication), προσωρινή αποθήκευση (caching) και συνεδρίες (sessions).

2.4.1 Πως Δουλευει η Laravel

Σε αντίθεση με άλλα περιβάλλοντα, το Laravel είναι μοναδικό με τον τρόπο που δίνει προτεραιότητα στη σύμβαση έναντι της διαμόρφωσης. Το Laravel χρειάζεται μόνο μερικές γραμμές κώδικα PHP για επεξεργασία και γίνεται έτοιμο για χρήση. Η αποφυγή ή η χρήση ενός ελάχιστου αριθμού αρχείων διαμόρφωσης δίνει σε όλες τις εφαρμογές ιστού του Laravel παρόμοια δομή κώδικα που είναι πολύ χαρακτηριστική και αναγνωρίσιμη. Αυτό μπορεί να θεωρηθεί με την πρώτη ματιά ως σοβαρός περιορισμός στον τρόπο με τον οποίο ένας προγραμματιστής μπορεί να επιθυμεί να οργανώσει τη δομή της δικής του εφαρμογής ιστού. Ωστόσο, αυτοί οι περιορισμοί καθιστούν πραγματικά πολύ πιο εύκολο να δημιουργήσετε εφαρμογές Ιστού.

2.4.2 Τι είναι το MVC

Ο όρος MVC [10] (Model, View, Controller) εκφράζει τον τρόπο με τον οποίο είναι δομημένη η διαδικτυακή εφαρμογή. Το MVC έγινε γρήγορα η ποιο συνηθισμένη πρακτική της βιομηχανίας για δημιουργία διαδικτυακών εφαρμογών και χρησιμοποιείται σε κάθε σύγχρονο περιβάλλον ανάπτυξης. Πολλά frameworks όπως Ruby on Rails (Ruby), ASP.NET (C#), CodeIgniter (PHP) το χρησιμοποιούν για να διαχωρίσουν τη λογική της εφαρμογής από το επίπεδο της αναπαράστασης. Ένα σχέδιο αρχιτεκτονικής MVC επιτρέπει στην εφαρμογή ιστού να έχει πολλές διαφορετικές προβολές ενός κοινού μοντέλου. Για παράδειγμα, σε ένα Eshop μπορεί να έχουμε πολλές προβολές, όπως η προβολή λίστας προϊόντων ή η προβολή γκαλερί προϊόντων. οπότε θα δημιουργηθεί ένα μοντέλο, δηλαδή ουσιαστικά ένας πίνακας κατηγοριών και μέσω αυτού του μοντέλου μπορούν να δημιουργηθούν πολλαπλές προβολές.

2.4.3 Μοντέλο (Model)

Ένα μοντέλο είναι ο τρόπος με τον οποίο ο προγραμματιστής μπορεί να χειριστεί τα δεδομένα. Αποτελείται από ένα επίπεδο που βρίσκεται μεταξύ των δεδομένων και της

εφαρμογής. Τα ίδια τα δεδομένα μπορούν να αποθηκευτούν σε διάφορους τύπους συστημάτων βάσεων δεδομένων όπως η MySQL ή PostgreSQL .

2.4.4 Προβολή (View)

Οι προβολές είναι η οπτική αναπαράσταση της εφαρμογής μας στο Web (επίπεδο παρουσίασης).Μπορεί να κατασκευαστεί εύκολα χρησιμοποιώντας τη γλώσσα προτύπου Blade που συνοδεύει Laravel που ουσιαστικά είναι HTML και CSS απλά με το πρότυπο Blade για να γίνει κατανοητή από την Laravel. Αυτό αλλιώς λέγεται και UI (User Interface) της εφαρμογής. Η προβολή (View) μιας εφαρμογής απαρτίζεται από γράμματα , κουμπιά, εικόνες, βίντεο, φόρμες συμπλήρωσης κτλπ, ότι μπορεί δηλαδή ο χρήστης να δει , να συμπληρώσει ή να πατήσει.

2.4.5 Συντονιστής (Controller)

Η κύρια λειτουργία ενός συντονιστή είναι ο χειρισμός αιτημάτων και η διαβίβαση δεδομένων από το μοντέλο στις Προβολές. Έτσι, ένας συντονιστής μπορεί να θεωρηθεί ως ο σύνδεσμος μεταξύ του Μοντέλου και της προβολής. Είναι η καρδιά του συστήματος δηλαδή το backend της σελίδας αυτό που δεν βλέπει ο χρήστης και αυτό που ουσιαστικά κάνει την σελίδα λειτουργική Είναι ο χειριστής εισόδου, όπως κλικ ή συμβάντα προγράμματος περιήγησης. Σκοπός του Controller είναι να ενημερώσει το μοντέλο όταν είναι απαραίτητο, για παράδειγμα εάν ένας χρήστης αλλάξει τον τίτλο ενός άρθρου τότε ο Controller θα κάνει τις απαραίτητες αλλαγές και την βάση δεδομένων άλλα και στο να φέρει πίσω στην σελίδα τον καινούργιο τίτλο.

2.4.6 Δρομολογητές (Routes)

Κάθε φορά που κάνουμε ένα αίτημα για ένα URL για να πάρουμε κάποια δεδομένα από τον Server , το route είναι αυτό που είναι υπεύθυνο να προωθήσει το αίτημα στον κατάλληλο controller. Υπάρχουν συνολικά δύο κατηγορίες routing παραμέτροι. Στην πρώτη και ποιο σημαντική κατηγορία ανήκει η υποχρεωτική route παράμετρος και στη δεύτερη ανήκει η Προαιρετική (optional) route παράμετρος. Στην περίπτωση που έχουμε πολλά παρόμοια routes έχουμε την δυνατότητα να δημιουργήσουμε ένα γενικό route. Αυτό γίνεται βάζοντας την παράμετρο ως μεταβλητή στη συνάρτηση ώστε να έχουμε ένα συγκεκριμένο αποτέλεσμα. Σαν αποτέλεσμα έχουμε τη μεταβλητή που είχαμε βάλαμε ως είσοδο τώρα την έχουμε ως έξοδο στο return. Τελος υπάρχει η δυνατότητα μια route παράμετρος γίνεται προαιρετική (optional) θέτοντας στο τέλος της το αγγλικό ερωτηματικό (?).

2.4.7 Query Builder

Το Query Builder είναι ένα εργαλείο που περιέχει μεθόδους αντίστοιχες με τις εντολές της MySQL και έτσι μας επιτρέπει την επικοινωνία της εφαρμογής με την βάση δεδομένων με απλούστερο και πολύ πιο ασφαλές τρόπο από το να γράφουμε τα queries σε SQL γλωσσά Χρησιμοποιώντας το εργαλείο αυτό κερδίζουμε χρόνο και κανουμε τον κώδικα ποιο ευανάγνωστο.

Παράδειγμα κώδικα που εκτελεί ερώτημα για το όνομα της βάσης δεδομένων.

```
$name = DB::Connection()->getDatabaseName();
```

2.4.8 Migrations & Schema Builder

Με τα migrations κατασκευάζουμε με εύκολο και γρήγορο τρόπο πίνακες στη Βάση Δεδομένων. εμείς γράφουμε τον κώδικα σε μορφή γλώσσας προγραμματισμού PHP και στην συνέχεια ο builder αναλαμβάνει να μετατρέψει αυτές της εντολές σε εντολές κώδικα SQL και να γίνουν οι απαραίτητες τροποποιήσεις στην βάση δεδομένων.

2.4.9 Laravel Authentication & Sessions

Σε αυτήν την ενότητα θα εξηγηθεί το πώς λειτουργεί η αυθεντικοποίηση (authentication) στο Laravel. Το σύστημα ελέγχου ταυτότητας (authentication system) είναι ουσιαστικά ένα ενδιάμεσο λογισμικό (middleware) που εξετάζει τους πόρους της αίτησης (cookies, request headers, session) για τα διαπιστευτήρια πρόσβασης (όνομα χρήστη και κωδικό πρόσβασης κ.λπ.). Η λογική εδώ είναι ότι ο χρήστης κατά την είσοδο στην εφαρμογή αποκτάει ένα μοναδικό κλειδί το οποίο διατηρείται στον περιηγητή (browser) για όσο ο χρήστης έχει ανοικτή αυτήν την σελίδα . Επίσης μπορούμε να επεκτείνουμε τον χρόνο ζωής αυτού του κλειδιού ώστε ακόμα και αν ο χρήστης αφήσει την σελίδα και την κλείσει αυτό το κλειδί να συνεχίσει να είναι αποθηκευμένο στον περιηγητή (browser) για κάποιο πεπερασμένο χρονικό διάστημα ανεξάρτητα αν είναι στην σελίδα η όχι. Με αυτόν τον τρόπο διευκολύνουμε τον χρήστη ώστε την επόμενη φορά που επισκεφθεί την σελίδα να μην χρειαστεί να ξανά βάλει τα στοιχειά του για να εισέλθει στην εφαρμογή αλλά έχει κατευθείαν πρόσβαση προσπερνώντας αυτό το βήμα διότι το κλειδί του είναι επικαιροποιημένο και είναι ενεργό και αποθηκευμένο στον περιηγητή (browser). Στην πραγματικότητα, υπάρχουν δύο ενδιάμεσα λογισμικά (middleware) που λειτουργούν για σκοπούς ελέγχου ταυτότητας. Το πρώτο είναι το Authenticate, το οποίο αποτελεί το κύριο γρανάζι του μηχανισμού ελέγχου ταυτότητας, και το ενδιάμεσο λογισμικό Authenticated Session. Αν δούμε την προτεραιότητα του middleware στο φακελο app → Http → Kernel.php, θα δούμε ότι το Authenticate βρίσκεται ακριβώς πριν από το Authenticated Session. Αυτός ο πίνακας ορίζει τη σειρά με την οποία το middleware προηγείται του άλλου. Έτσι αποφεύγεται η σύγκρουση όταν ένα middleware χρειάζεται αυτό που παρέχει ένα άλλο. Όταν ορίζετε το ενδιάμεσο λογισμικό auth στον δρομολογητή (router) το εισερχόμενο αίτημα θα χειρίζεται η κλάση Authenticate. Αυτό

το middleware θα προσπαθήσει να ελέγξει τον προεπιλεγμένο μηχανισμό ελέγχου ταυτότητας, που έχει οριστεί στο αρχείο ρυθμίσεων, διαφορετικά θα κάνει κύκλο κάθε έλεγχο ταυτότητας που έχετε ορίσει στις παραμέτρους του middleware . Όταν διατρέχει με τη σειρά κάθε φρουρό ελέγχου ταυτότητας, ουσιαστικά καλεί τη μέθοδο check() κάθε φρουρού μέσα στον AuthManager, επιστρέφοντας ένα boolean. Η εφαρμογή θα χρησιμοποιήσει τον πρώτο έλεγχο ταυτότητας που επιστρέφει true, και αν κανένας δεν λέει ότι υπάρχει ένας χρήστης με έλεγχο ταυτότητας στη θέση του, θα επιστρέψει ένα AuthenticationException επειδή δεν υπήρξε έλεγχος ταυτότητας ικανός να πει ποιος έχει πρόσβαση στη διαδρομή. Η εξαίρεση (Exception) θα λάβει τη διεύθυνση URL στην οποία θέλουμε να ανακατευθύνουμε τον χρήστη εάν δεν έχει πιστοποιηθεί μέσω της μεθόδου redirectTo(), η οποία λαμβάνει το Request ως παράμετρο. Θα μπορούσατε, για παράδειγμα, να ανακατευθύνετε τον Χρήστη σε διαφορετική διεύθυνση URL ανάλογα με το πού έχει γίνει η Αίτηση. Αυτό το ενδιάμεσο λογισμικό είναι απενεργοποιημένο από προεπιλογή, αλλά η ενεργοποίησή του θα επιτρέψει στο χρήστη να αποσυνδεθεί από κάθε άλλη συσκευή όταν αλλάζει ο κωδικός πρόσβασής του. Μια γρήγορη ματιά στη μέθοδο handle() μας λέει ότι αυτή η AuthenticateSession προσπαθεί να επικυρώσει την αυθεντικοποίηση του χρήστη συγκρίνοντας το hash του κωδικού πρόσβασης που είχε αποθηκευτεί προηγουμένως. Έτσι, αν ο χρήστης αλλάξει το hash του κωδικού πρόσβασης στη βάση δεδομένων, όλες οι άλλες συνεδρίες θα ακυρωθούν αυτόματα επειδή αυτές χρησιμοποιούν το παλιό hash του κωδικού πρόσβασης. Σε περίπτωση που δεν το γνωρίζατε, το Laravel δεν θα αναγκάσει τον χρήστη να πιστοποιηθεί μόνο σε μία συσκευή. Αυτό σημαίνει ότι ένας χρήστης μπορεί να έχει πολλαπλές συνεδρίες ανοιχτές σε διαφορετικά μέρη του κόσμου, ακόμη και αν ο χρήστης αλλάξει τον κωδικό πρόσβασής του. Για να αντιμετωπιστεί αυτό, το Laravel επιτρέπει τη χρήση αυτού του ενδιάμεσου λογισμικού σε συνδυασμό με τη μέθοδο logoutOtherDevices(), η οποία θα ακυρώσει τη σύνοδο στις άλλες συσκευές όταν ο χρήστης συνδεθεί. Αυτό μπορεί να τοποθετηθεί στον LoginController σας. Αυτό γινεται ως εξής, όταν λαμβάνει τον κωδικό πρόσβασης, τον επανακαθαρίζει και τον αποθηκεύει στη βάση δεδομένων, ακυρώνοντας έτσι τις άλλες συνεδρίες, αφού το hash δεν θα είναι ίσο.

2.4.10 Laravel validation

Η επικύρωση Laravel μπορεί να πραγματοποιηθεί με διάφορους τρόπους και τα μηνύματα σφάλματος παράγονται είτε αυτόματα είτε χειροκίνητα, ανάλογα με τη μέθοδο επικύρωσης της επιλογής σας. Μόλις επικυρωθεί η είσοδος, τα υπόλοιπα εξελίσσονται όπως αναμενόταν, αυτόματα. Με αυτόν τον τρόπο αποφεύγουμε περαιτέρω σφάλματα στην πορεία.

• Συγγραφή της λογικής επικύρωσης
Η μέθοδος validate που βρίσκεται στο αντικείμενο Illuminate → Http → Request, είναι ένα από τα αντικείμενα του πίνακα επικύρωσης του Laravel. Σε περίπτωση αποτυχίας της επικύρωσης, το πλαίσιο δημιουργεί μια αυτόματη απάντηση για τον χρήστη στο αρχείο Illuminate → Validation → ValidationException. Αν όλα είναι καλά, η εκτέλεση

συνεχίζεται κανονικά. Η επικύρωση Laravel σταματάει στο πρώτο σφάλμα. Σε ορισμένες περιπτώσεις, η διακοπή στο πρώτο σφάλμα επικύρωσης μπορεί να απαιτείται για την τιμή των μεταβλητών.

• Εμφάνιση σφαλμάτων επικύρωσης
Έχετε ορίσει τους κανόνες επικύρωσης στο Laravel και η είσοδος χρήστη που λαμβάνετε δεν συμμορφώνεται. Αυτό που κάνει η επικύρωση του Laravel είναι να μεταφέρει τον χρήστη πίσω στην προηγούμενη σελίδα και το κάνει αυτόματα. Τα σφάλματα εισόδου αίτησης και επικύρωσης εμφανίζονται επίσης αυτόματα. Εάν εφαρμόσετε το middleware Illuminate → View → Middleware → ShareErrorsFromSession, μια μεταβλητή $errors μοιράζεται σε όλες τις προβολές της εφαρμογής, ώστε να ξέρετε ότι είναι πάντα καθορισμένη. Αυτή η μεταβλητή επικύρωσης του Laravel μπορεί να βρεθεί στην Illuminate → Support → MessageBag ως παράδειγμα.

• Επικύρωση αιτήματος φόρμας στο Laravel
Η επικύρωση αίτησης φόρμας χρησιμοποιείται για σενάρια επικύρωσης αυξημένης πολυπλοκότητας. Αυτές οι κλάσεις αίτησης φόρμας είναι προσαρμοσμένες κλάσεις με δική τους λογική επικύρωσης και εξουσιοδότησης. Για να δημιουργήσετε την κλάση αίτησης, εκτελέστε την εντολή PHP Artisan make:request. Η κλάση αίτησης φόρμας που δημιουργείται με αυτόν τον τρόπο βρίσκεται στον κατάλογο app → Http → Requests. Σε περίπτωση αποτυχίας της επικύρωσης, ο χρήστης οδηγείται πίσω στην προηγούμενη τοποθεσία με μια απάντηση αυτόματης ανακατεύθυνσης. Στην περίπτωση ενός αιτήματος XHR, ο χρήστης λαμβάνει τον κωδικό κατάστασης 422 της απόκρισης HTTP με την αναπαράσταση JSON των σφαλμάτων επικύρωσης. Για να δουμε αν ένας συγκεκριμένος χρήστης είναι πραγματικά εξουσιοδοτημένος υπάρχει η μέθοδος authorize που είναι διαθέσιμη σε αυτή την κλάση αίτησης φόρμας. Για να ελέγξετε τις πληροφορίες του πιστοποιημένου χρήστη, μπορείτε να εκτελέσετε τη μέθοδο user, καθώς η βασική κλάση αίτησης του Laravel επεκτείνεται από τις αιτήσεις φόρμας. Μια ψευδής επιστροφή της μεθόδου authorize μεταφράζεται σε κωδικό κατάστασης 403 και δεν θα υπάρξει εκτέλεση της μεθόδου του ελεγκτή σας. Επιστρέψτε true στη μέθοδο authorize αν επιλέξετε να χειριστείτε τη λογική εξουσιοδότησης σε άλλα μέρη της εφαρμογής.

• Μηνύματα σφάλματος
Το Laravel παρέχει μια ποικιλία μεθόδων για μηνύματα σφάλματος. Είναι διαθέσιμες στην περίπτωση Illuminate → Support → MessageBag, την οποία λαμβάνετε σε μια περίπτωση επικυρωτή όταν καλείτε τη μέθοδο errors. Μπορούμε να πάρουμε τα πρώτα ή όλα τα μηνύματα για ένα πεδίο (η μέθοδος first ή η μέθοδος get). Στην τελευταία περίπτωση, μπορούμε να χρησιμοποιήσουμε τον χαρακτήρα * για να ανακτήσετε όλα τα μηνύματα για κάθε πίνακα πεδίου φόρμας.

2.4.11 Seeding

Αρκετές φορές θα χρειαστεί να δοκιμάσουμε την λειτουργικότητα της βάσης δεδομένων μας καθώς και διάφορες λειτουργίες που ενδεχομένως να έχει. Για να το κάνουμε αυτό χρησιμοποιούμε τους seeders οι οποίοι είναι υπεύθυνοι να γεμίσουν τον πινάκα με κάποιες ενδεικτικές τιμές που ταιριάζουν στο τύπο του κάθε πεδίου από την βάση δεδομένων και με αυτό τον τρόπο αυτοματοποιούμε αυτήν την διαδικασία καθώς μπορεί να γίνει σε επανάληψη και για μεγάλο όγκο δεδομένων αν αυτό χρειαστεί και χωρίς να υπάρχει η ανάγκη να γραφτεί κάποιος πολύπλοκος κώδικας SQL. Η διαδικασία αυτή ονομάζετε seeding.

2.4.12 Eloquent ORM

Με το Eloquent δημιουργείται μια γέφυρα μεταξύ του Model (π.χ. Article Model) και του πίνακα στη βάση δεδομένων (π.χ. articles table). Eloquent ORM είναι το ακρωνύμιο των λέξεων Object Relational Mapping και χρησιμοποιείται για να αλληλοεπιδρά με σχεσιακές Βάσεις Δεδομένων. Όταν χρησιμοποιούμε το Eloquent, τότε το όνομα του Model θα πρέπει να είναι στον ενικό αριθμό και το όνομα του πίνακα στον πληθυντικό αριθμό. Η χρησιμότητα του είναι να μετατρέψει εντολές PHP γραμμένες με σκοπό την αλληλεπίδραση και την επικοινωνία με την βάση σε εντολές SQL για να τις καταλάβει η βάση δεδομένων. για παράδειγμα όταν θέλουμε να μας επιστρέψει ένα πίνακα η βάση δεδομένων το ORM θα πάρει τον κώδικα από το αρχείο PHP που του δώσαμε και θα το μετατρέψει σε SQL κώδικα ώστε η βάση να καταλάβει το ερώτημα και να επιστρέψει το σωστό αποτέλεσμα.

2.4.13 Eloquent: Serialization

Κατά την αποστολή ενός αντικειμένου στην ουρά, στο παρασκήνιο το Laravel σειριοποιεί αναδρομικά το αντικείμενο και όλες τις ιδιότητές του σε μια αναπαράσταση συμβολοσειράς που στη συνέχεια γράφεται στην ουρά. Εκεί περιμένει έναν εργάτη ουράς για να το ανακτήσει από την ουρά και να το αποσειράρει πίσω σε ένα αντικείμενο PHP. Όταν τα περίπλοκα αντικείμενα σειριοποιούνται, οι αναπαραστάσεις τους με συμβολοσειρές μπορεί να είναι φρικτά μεγάλες, καταλαμβάνοντας περιττούς πόρους τόσο στην ουρά όσο και στους διακομιστές εφαρμογών. Εξαιτίας αυτού, το Laravel προσφέρει μια ιδιότητα που ονομάζεται SerializesModels η οποία, όταν προστίθεται σε ένα αντικείμενο, βρίσκει οποιεσδήποτε ιδιότητες τύπου Model ή Eloquent\Collection κατά τη σειριοποίηση και τις αντικαθιστά με ένα απλό PHP αντικείμενο γνωστό ως ModelIdentifier. Αυτά τα αντικείμενα αναπαριστούν τις αρχικές ιδιότητες Model type και ID, ή IDs στην περίπτωση μιας Eloquent\Collection, με μια πολύ μικρότερη αναπαράσταση συμβολοσειράς κατά τη σειριοποίηση. Όταν αυτά τα αντικείμενα αποσειροποιούνται, τα ModelIdentifiers αντικαθίστανται στη συνέχεια με το Μοντέλο ή τη Συλλογή Μοντέλων Eloquent\Collection που αναπαριστούσαν προσωρινά.

Ας υποθέσουμε ότι έχουμε δύο πεδία στον πίνακα μας για τους χρήστες μας : όνομα και επώνυμο. Και παρόλο που αυτά τα δύο πεδία μπορούν να χρησιμοποιηθούν όταν

χρειάζεται, αλλά μερικές φορές χρειαζόμαστε αυτά τα δύο συνδυασμένα για να εμφανίσουμε το πλήρες όνομα ως ένα ενιαίο πεδίο, να έχουμε λειτουργίες σε αυτά στο σύνολό τους.

Ο Κλασσικός PHP τρόπος :

{{ $user->first_name . ' ' . $user->last_name }}

Ο τρόπος Laravel :

{{ $user->full_name }}

Έτσι, αυτό που συμβαίνει εδώ είναι ότι έχουμε δημιουργήσει μια συνάρτηση η οποία θα καλείται κάθε φορά που το full_name καλείται στο user(Model). Υπάρχει ένα μοτίβο, αν το δούμε προσεκτικά, σε κάθε συνάρτηση που φτιάχνουμε. Αλλά υπάρχει μια παγίδα, δεν μπορούμε να χρησιμοποιήσουμε αυτό το όνομα του χαρακτηριστικού σε εύγλωττα ερωτήματα όπως οποιαδήποτε άλλα πεδία, τα εύγλωττα ερωτήματα λειτουργούν στα πεδία της βάσης δεδομένων.

2.4.14 Συσχετίσεις (Relations)

Οι συσχετίσεις που υπάρχουν σε όλες της διαδικτυακές εφαρμογές είναι οι εξής:

Χρησιμοποιώντας ένα τυχαίο παράδειγμα Συγγραφέα / βιβλία.

1) Ένα προς ένα (One to One): αυτό από την πλευρά του συγγραφέα σημαίνει ότι ένας συγγραφέας έχει ένα βιβλίο που έχει γράψει ο ίδιος, από την πλευρά του βιβλίου σημαίνει ότι ένα βιβλίο ανήκει σε έναν και μόνο συγγραφέα και όχι σε πολλούς δηλαδή ο συγγραφέας του βιβλίου είναι ένας. Σε αυτην την περίπτωση λέμε ότι έχουμε μια σχέση ένα προς ένα

2) Ένα προς πολλά (One to Many) **:** αυτό από την πλευρά του συγγραφέα σημαίνει ότι ένας συγγραφέας έχει πολλά βιβλία τα οποία έχει γράψει ο ίδιος και από την πλευρά των βιβλίων υπάρχουν πολλά βιβλία τα οποία ανήκουν σε έναν και μόνο συγγραφέα.

3) Πολλά προς πολλά (Many to Many) **:** αυτό από την πλευρά του συγγραφέα σημαίνει ότι πολλοί συγγραφείς έχουν γράψει πολλά βιβλία και από την πλευρά των βιβλίων υπάρχουν πολλά βιβλία τα οποία ανήκουν σε πολλούς και μόνο συγγραφέα.

2.4.15 Composer & Artisan CLI

Ο Composer είναι ένα εργαλείο για να διαχειριζόμαστε τα διάφορα πακέτα της PHP. Η κύρια λειτουργία του Composer είναι να κατεβάσει και να εγκαταστήσει καθώς και να ενημερώσει τα PHP πακέτα που χρησιμοποιούμε στην εφαρμογή μας. Το Artisan Command-Line Interface (CLI) είναι ένα βοηθητικό πρόγραμμα γραμμής εντολών (Command-Line) μέσω του οποίου χειριζόμαστε το περιβάλλον του project μας και είναι ενσωματωμένο στο Laravel Framework. Έτσι, ο προγραμματιστής χρησιμοποιώντας το Artisan CLI αλληλοεπιδρά με το Laravel Framework. Με το Artisan μπορούμε να γράψουμε εντολές που δημιουργούν τους Controllers, τα Models, τα Views καθώς και εντολές που εκτελούν κάποια κομμάτια κώδικα.

2.4.16 Laravel Scopes

Τι είναι τα πεδία εφαρμογής (Laravel Scopes); Ένα πεδίο εφαρμογής είναι μια μέθοδος στο μοντέλο που καθιστά δυνατή την προσθήκη λογικής βάσης δεδομένων στο μοντέλο . Τα πεδία εφαρμογής καθιστούν δυνατή την επαναχρησιμοποίηση της λογικής ερωτημάτων επιτρέποντας την ενθυλάκωση της λογικής της βάσης δεδομένων σε ένα μοντέλο.

Global Scopes: Τα Global Scopes επιτρέπουν να προσθέσουμε περιορισμούς σε όλα τα ερωτήματα ενός μοντέλου. Με αυτόν τον τρόπο εξασφαλίζετε ότι κάθε ερώτημα σε ένα δεδομένο μοντέλο έχει ορισμένους περιορισμούς. Μπορούμε να δημιουργήσετε ένα global scope εκτελώντας την ακόλουθη εντολή:
php artisan make:scope MessageScope
Αυτό θα δημιουργήσει την κλάση MessageScope στο φάκελο app → Scopes.

Ανώνυμα Global Scopes : Αν εχουμε global scopes που δεν χρειάζονται τη δική τους κλάση, μπορούμε να ορίσουμε ένα ανώνυμο global scope χρησιμοποιώντας ένα Closure. Αυτό γίνεται επίσης στη μέθοδο εκκίνησης του μοντέλου.

Αφαίρεση ενός global scope: Είναι δυνατή η αφαίρεση ενός global scope, εάν έχουμε ένα συγκεκριμένο ερώτημα που θέλουμε να εκτελέσουμε χωρίς το global scope . Μπορούμε να καλέσουμε τη μέθοδο withoutGlobalScope με παράμετρο το όνομα κλάσης της εμβέλειας και το global scope θα αφαιρεθεί.

Local Scopes: Τα Local Scopes εφαρμογής επιτρέπουν τον ορισμό κοινών περιορισμών που είναι εύκολα επαναχρησιμοποιήσιμα. Αυτό είναι χρήσιμο αν θέλουμε να λαμβάνουμε μόνο τα δημοσιευμένα μηνύματα, για παράδειγμα. Μετά τον ορισμό ενός ή περισσότερων Local Scopes εφαρμογής, αυτά μπορούν να χρησιμοποιηθούν καλώντας τη μέθοδο scope στο μοντέλο. Είναι δυνατή η αλυσιδωτή σύνδεση μεθόδων εμβέλειας.
Dynamic Local Scopes : Τα Dynamic Local Scopes λειτουργούν με τον ίδιο ακριβώς τρόπο όπως τα κανονικά Local Scopes. Η μόνη διαφορά είναι ότι ενα Dynamic Local Scope δέχεται παραμέτρους.

2.5 JavaScript Βιβλιοθήκες

Οι βιβλιοθήκες JavaScript είναι συλλογές προ-γραμμένων αποσπασμάτων κώδικα που μπορούν να χρησιμοποιηθούν και να επαναχρησιμοποιηθούν για την εκτέλεση κοινών λειτουργιών JavaScript. Ένας συγκεκριμένος κώδικας βιβλιοθήκης JavaScript μπορεί να συνδεθεί στον υπόλοιπο κώδικα του προγράμματος. Αυτό έχει ως αποτέλεσμα την γρηγορότερη ανάπτυξη του προγράμματος και μειώνει σημαντικά τα προγραμματιστικά λάθη. Υπάρχουν πολλές βιβλιοθήκες και διαθέσιμες στους προγραμματιστές JavaScript σήμερα, και παρακάτω θα δούμε αυτές που έχουν χρησιμοποιηθεί σήμερα για την ανάπτυξη της εφαρμογής .

2.5.1 Axios

Το Axios είναι η ποιο γνωστή βιβλιοθήκη της JavaScript για να καλέσει τον διακομιστή (Server) από το γραφικό περιβάλλον της εφαρμογής (User interface) με σκοπό να αντλήσει τα δεδομένα που χρειάζεται ώστε να τα εμφανίσει τον εκάστοτε χρήστη. Ουσιαστικά είναι ο τρόπος με τον οποίο γίνεται άμεση η επικοινωνία μεταξύ του διακομιστή (Server) με το γραφικό περιβάλλον της εφαρμογής (UI). Θεωρείτε απαραίτητο σε μεγάλου μεγέθους εφαρμογές διότι εκεί σίγουρα θα υπάρχει ένας διακομιστής και μια βάση δεδομένων ώστε να χρειαστεί να έρθουν δεδομένα η να τροποποιηθούν η να διαγραφτούν η και να δημιουργηθούν καινούργια.

2.5.2 Bootstrap

Το Bootstrap είναι ένα δωρεάν και ανοιχτού κώδικα εργαλείο ανάπτυξης ιστοσελίδων. Έχει σχεδιαστεί για να διευκολύνει τη διαδικασία ανάπτυξης ιστοσελίδων, παρέχοντας μια ποικιλία από έτοιμα και μορφοποιημένα κομμάτια κώδικα. Με άλλα λόγια, το Bootstrap βοηθά τους προγραμματιστές ιστού να κατασκευάζουν ταχύτερα ιστότοπους, καθώς επιταχύνει την διαδικασία και ταυτόχρονα κάνει την ιστοσελίδα να ανταποκρίνεται και σε κινητές συσκευές εκτός από τον υπολογιστή.

2.5.3 SaSS

Το Sass (το οποίο σημαίνει "Syntactically awesome style sheets") είναι μια επέκταση του CSS που μας επιτρέπει να χρησιμοποιούμε ποιο σύνθετα πράγματα όπως μεταβλητές, ένθετους κανόνες, inline εισαγωγές και πολλά άλλα. Βοηθά επίσης να διατηρείτε την μορφοποίηση της εφαρμογής οργανωμένη και μας επιτρέπει να γραφουμε το style της εφαρμογής ποιο γρήγορα και ποιο οργανωμένα .Το Sass είναι συμβατό με όλες τις εκδόσεις της CSS.

2.6 Περιβάλλον προγραμματισμού

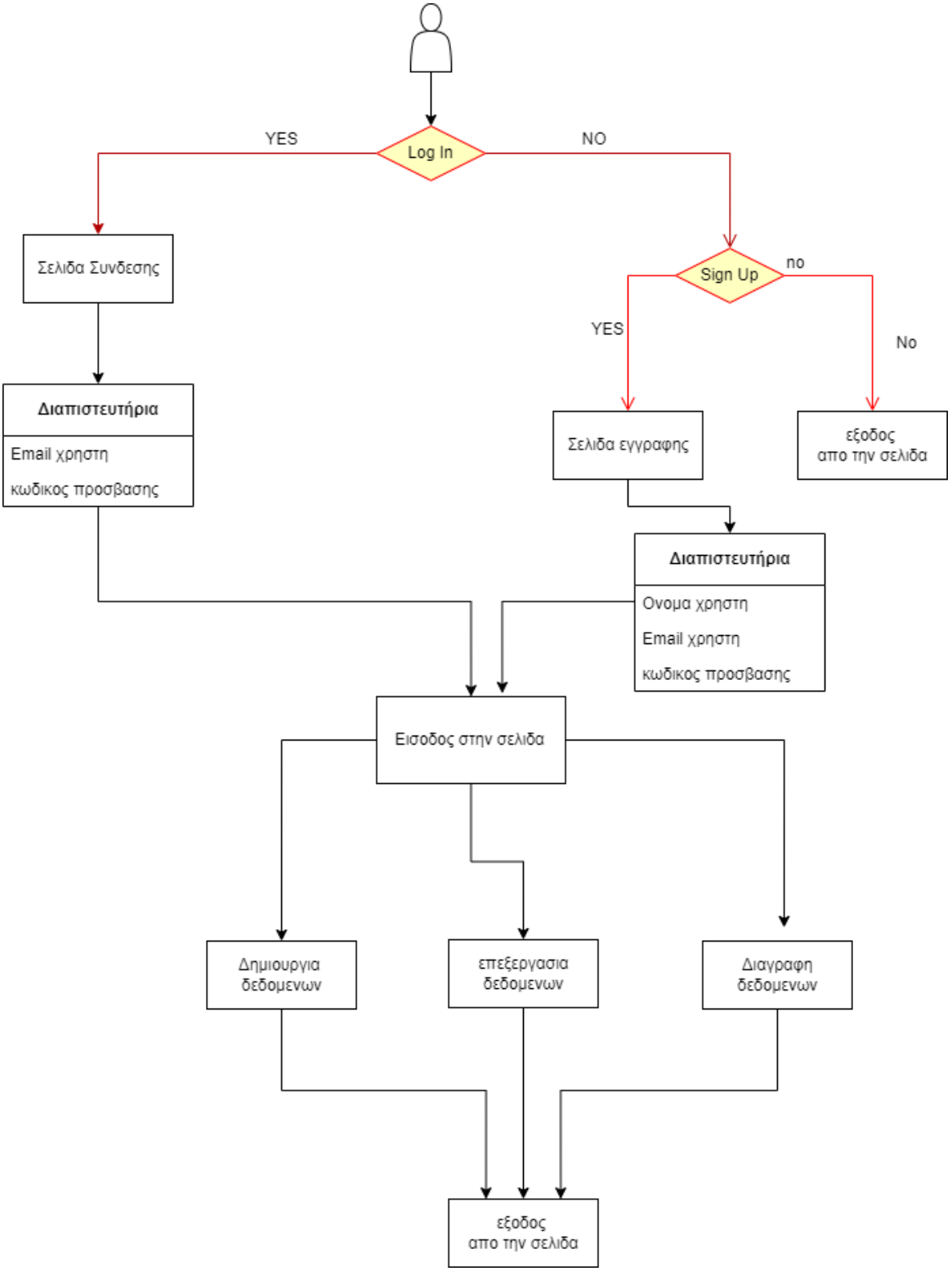
Το περιβάλλον προγραμματισμού η αλλιώς development environment είναι το περιβάλλον στο οποίο δημιουργείτε και αναπτύσσετε η εφαρμογή. Όλες αυτές οι τεχνολογίες , βιβλιοθήκες και γλώσσες προγραμματισμοί που έχουν αναφερθεί

συνθέτουν μαζί ένα περιβάλλον προγραμματισμοί για την ανάπτυξη μιας εφαρμογής. Ένα από τα ποιο διάσημα τέτοια εργαλεία που συνθέτουν ένα περιβάλλον προγραμματισμού είναι το XAMPP. Αυτό το εργαλείο έχει χρησιμοποιηθεί για την ανάπτυξη της συγκεκριμένης εφαρμογής καθώς περιεχέι ένα ολοκληρωμένο περιβάλλον με την γλωσσά προγραμματισμού PHP , μια βάση τύπου MySQL την MariaDB η οποία είναι η Open Source έκδοση της MySQL και έναν Apache HTTP Server για να μπορέσει να σηκωθεί η εφαρμογή σε κάποιον περιηγητή.

3. Ανάλυση Σχεδίασης του Συστήματος

3.1 Διάγραμμα αρχιτεκτονικής

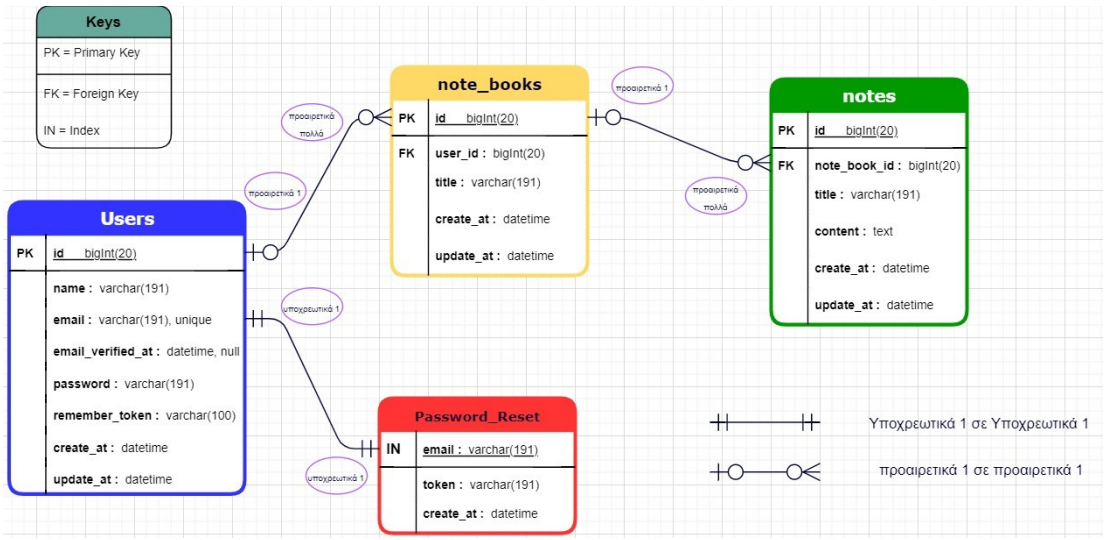
Σε αυτήν την ενότητα θα γίνει ανάλυση και επεξήγηση διαγραμμάτων όσον άφορα την αρχιτεκτονική της εφαρμογής και την δομή της βάσης δεδομένων. Η παρακάτω φωτογραφία είναι μια αναπαράσταση ενός χρήστη που χρησιμοποιεί την εφαρμογή. Ως χρήστης λοιπόν αποικίζεται το ανθρωπάκι τέρμα πάνω, έτσι ο χρήστης όταν ανοίξει την εφαρμογή η 1η επιλογή που έχει να κάνει είναι να διαλέξει αν θέλει να κάνει είσοδο στην εφαρμογή με το email του και τον κωδικό πρόσβασης , με την προϋπόθεση ότι έχει ήδη έναν λογαριασμό. Αν δεν έχει λογαριασμό του δίνεται η δυνατότητα να δημιουργήσει έναν βάζοντας ένα όνομα, ένα email και έναν κωδικό πρόσβασης. Τα επόμενα βήματα αφορούν μόνο χρήστες που έχουν κάνει είσοδο στην εφαρμογή είτε είναι από εγγραφή είτε από είσοδο με υπάρχων λογαριασμό. Μετά την είσοδο στην εφαρμογή ο κάθε χρήστης ξεχωριστά έχει την δυνατότητα να χρησιμοποιήσει τις λειτουργίες της με οποίον τρόπο θέλει δηλαδή μπορεί να δημιουργήσει νέες σημειώσεις , να επεξεργαστεί ήδη υπάρχουσες σημειώσεις η να διαγράψει μέρος αυτόν η και όλες. Εφόσον επιλέξει να κάνει αυτές τις ενέργειες και δεν επιθυμεί κάτι περισσότερο μπορεί να κάνει αποσύνδεση από τον λογαριασμό του και να επιστρέψει ξανά όποια άλλη στιγμή θελήσει.



Εικόνα 10 Δομή αρχιτεκτονικής της Εφαρμογής

3.2 Ανάλυση της Δομής της βάσης δεδομένων

Στην παρακάτω εικόνα φαίνεται η δομή της Βάσης δεδομένων . Η λογική που έχει φτιαχτεί είναι ότι ο χρήστης έχει κάποια πεδία όπως φαίνεται και στο διάγραμμα και όταν είναι συνδεδεμένος στην εφαρμογή μπορεί να δημιουργήσει ένα , κανένα ή περισσότερα σημειωματάρια (notebooks). Το κάθε σημειωματάριο (notebook) αναπαριστά μια κατηγορία από σημειώσεις. Κάθε φορά που φτιάχνεται μια τέτοια κατηγορία ένα από τα πεδία αυτού του αντικείμενου είναι και το id του χρήστη έτσι ώστε να μπορεί να διαφοροποιηθεί από τις υπόλοιπες κατηγορίες και να ανήκει μοναδικά σε έναν και μόνο χρήστη. Η συσχέτιση που έχουν μεταξύ τους αυτοί οι δυο πίνακες είναι ένα προς πολλά δηλαδή κάθε χρήστης μπορεί να έχει δημιουργήσει πολλές κατηγορίες αλλά κάθε κατηγορία ανήκει σε έναν και μόνο χρήστη. Με ποιο απλά λόγια ο χρήστης έχει ένα , κανένα η πολλά σημειωματάρια και από την άλλη κάθε σημειωματάριο ανήκει σε έναν μοναδικό χρήστη. Στην συνεχεία το κάθε σημειωματάριο αποτελείται από πολλές σημειώσεις έτσι λοιπόν ο πινάκας notes περιεχέι τις σημειώσεις οι οποίες ανήκουν σε κάποιο συγκεκριμένο σημειωματάριο ενός χρήστη, με ποιο απλά λόγια δηλαδή κανένας δεν έχει πρόσβαση σε καμιά σημείωση άλλου χρήστη. Όπως βλέπουμε και στην εικόνα το κάθε σημείωμα (note) έχει και ένα πεδίο με το id από το σημειωματάριο (notebook) στο οποίο ανήκει. Έτσι λοιπόν συμπεραίνουμε ότι η σχέση ανάμεσα στους δυο πίνακες (notebook) και (note) είναι ένα προς πολλά δηλαδή ένα σημειωματάριο (notebook) μπορεί να έχει ένα κανένα η πολλές σημειώσεις ενώ από την άλλη πλευρά μια σημείωση ανήκει σε έναν και μόνο σημειωματάριο και σε έναν και μόνο χρήστη.

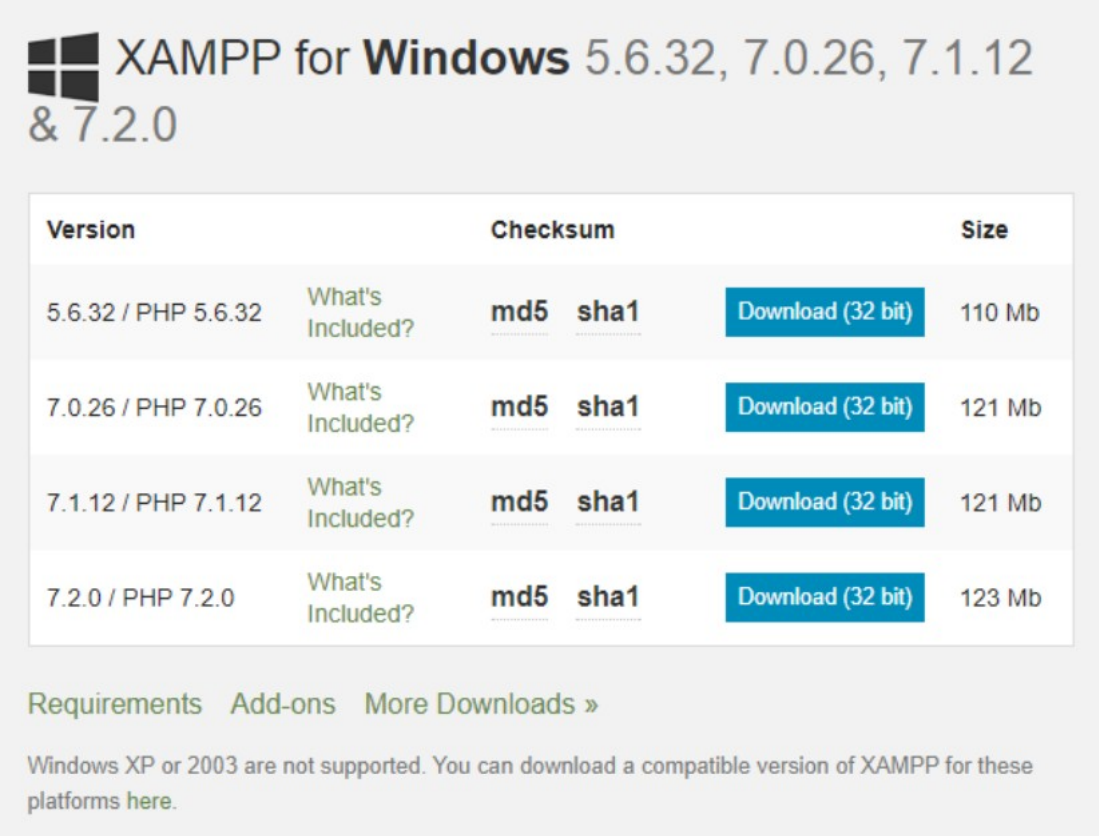


Εικόνα 11 Δομή της βάσης Δεδομένων

4. Υλοποίηση συστήματος

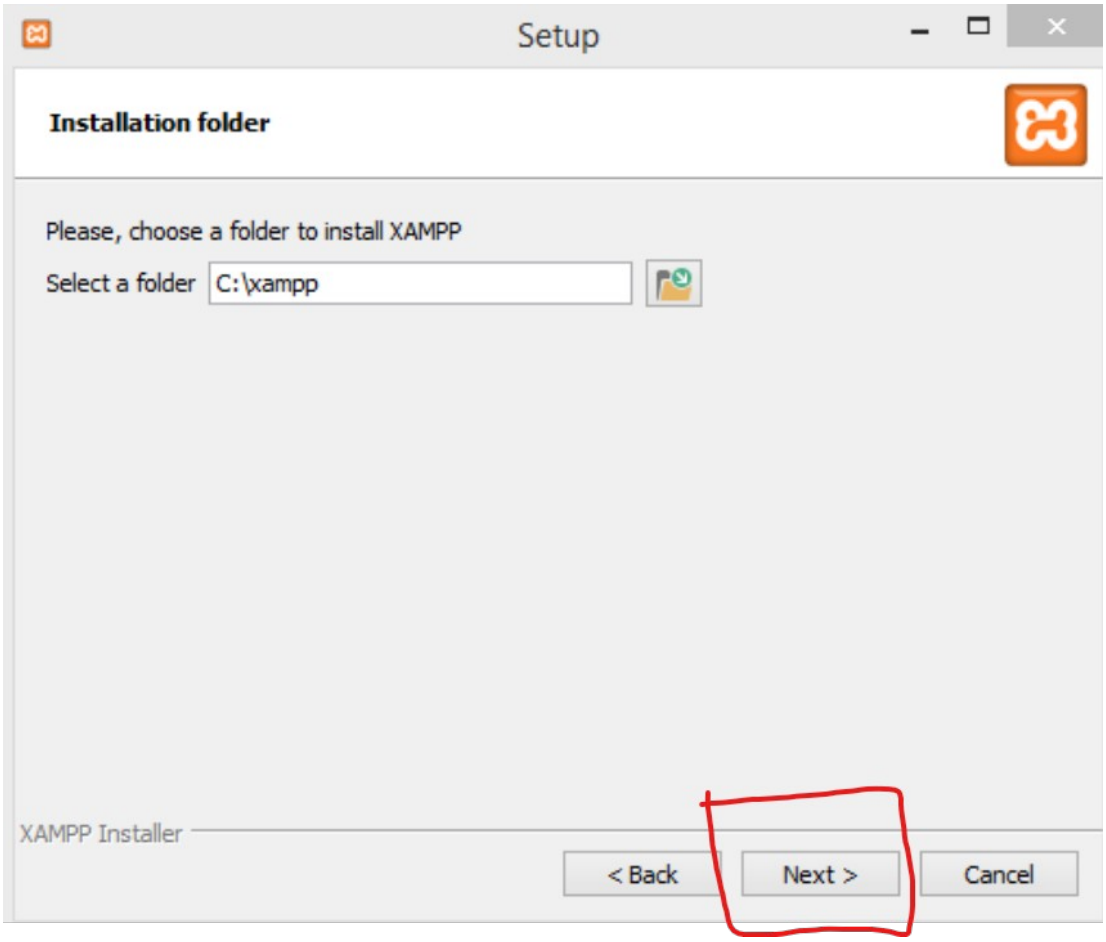
4.1 Εγκατάσταση του XAMPP

Βήμα 1: Πηγαίνουμε στο https://www.apachefriends.org/download.html και βρίσουμε τον installer που χρειαζόμαστε.



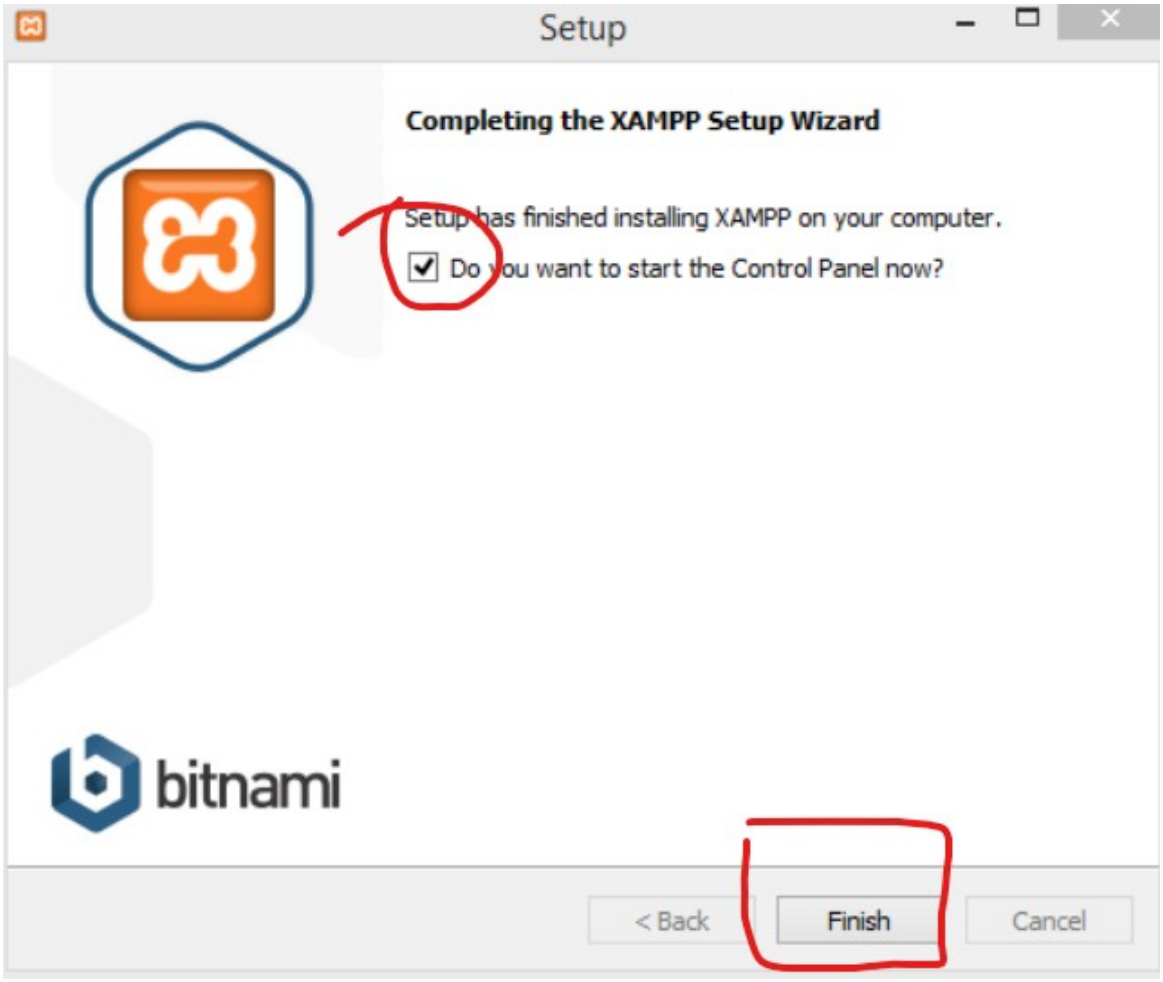
Εικόνα 12 Εγκατάσταση XAMPP

Βήμα 2: Βήμα 2: Τρέχουμε τον Installer και ξεκινάμε τη εγκατάσταση. Επιλέγουμε C:xampp σαν το directory εγκατάστασης και πατάμε next



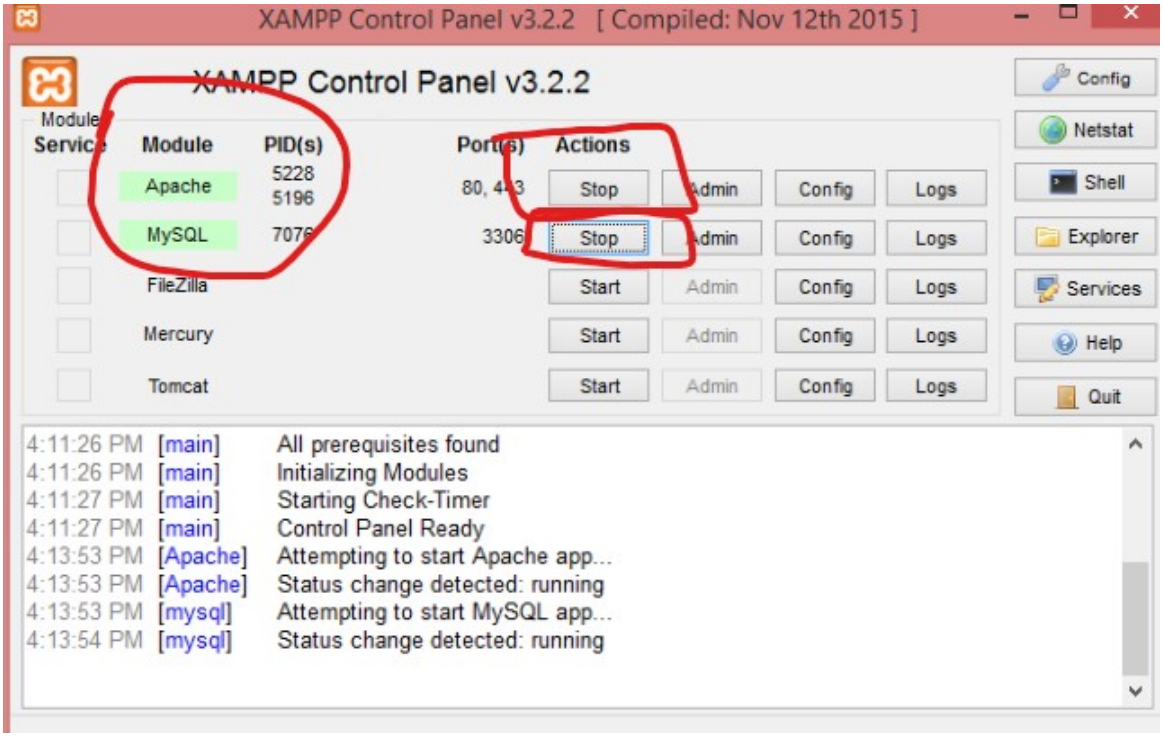
Εικόνα 13 Εγκατάσταση XAMPP

Βήμα 3: Δεν κάνουμε καμία αλλαγή στις επιλογές και πατάμε Install και περιμένουμε να ολοκληρωθεί η διαδικασία. Μόλις ολοκληρωθεί η εγκατάσταση θα δούμε την παρακάτω εικόνα.



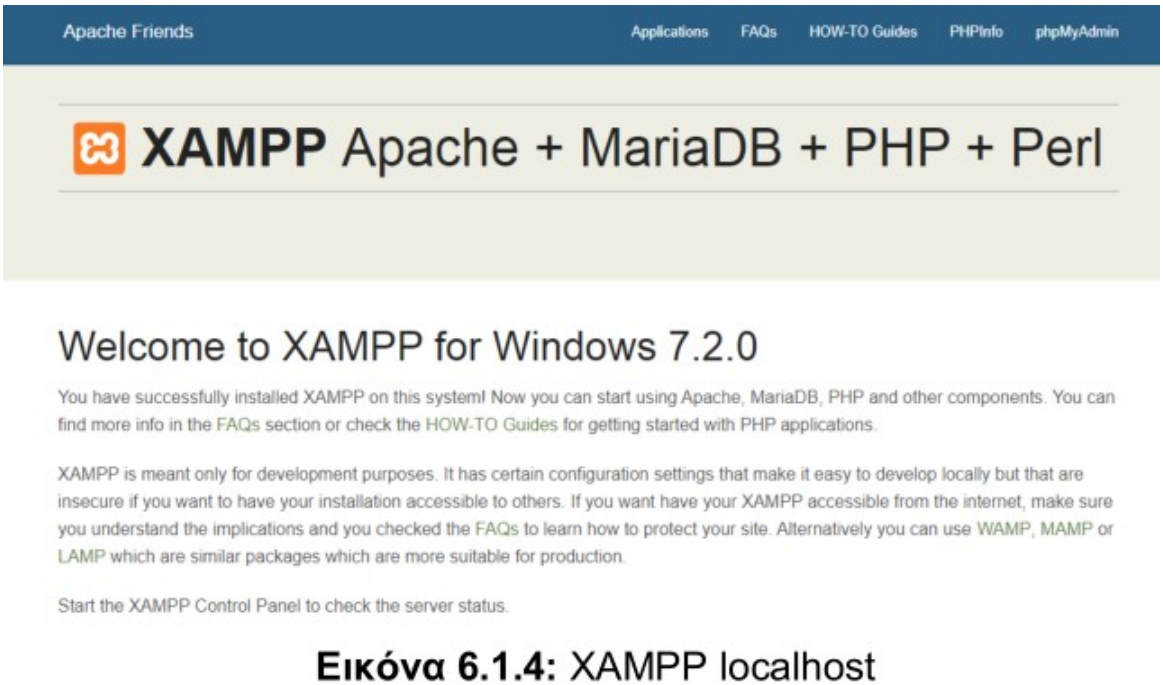
Εικόνα 14 Εγκατάσταση XAMPP

Βήμα 5: Ανοίγουμε το Control Panel του XAMPP και πατάμε start στις επιλογές Apache και MySQL.



Εικόνα 15 Εγκατάσταση XAMPP

Βήμα 6: Για να διαπιστώσουμε ότι όλα κύλισαν ομαλά με την εγκατάσταση μας, ανοίγουμε τον Browser μας και πληκτρολογούμε την διεύθυνση http://localhost αν μας δείτε την παρακάτω εικόνα τότε η εγκατάσταση ολοκληρώθηκε επιτυχώς.

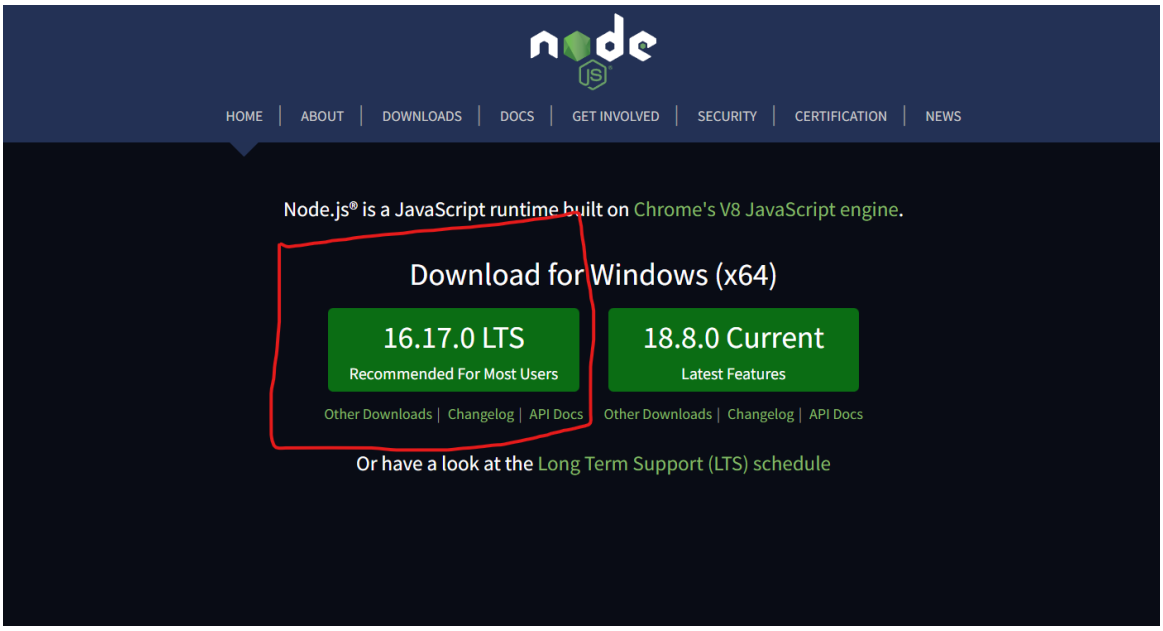


Εικόνα 6.1.4: XAMPP localhost

Εικόνα 16 Εγκατάσταση XAMPP

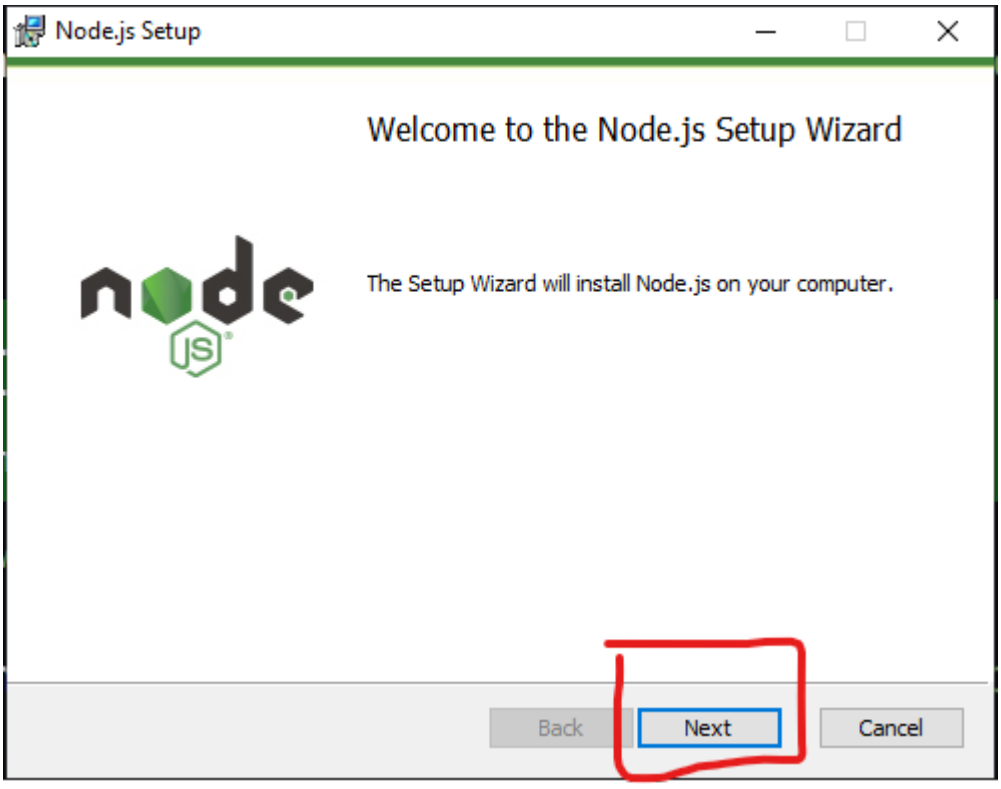
4.2 Εγκατάσταση NodeJS

Βήμα 1: Για να εγκαταστήσουμε επιτυχώς το NodeJS πηγαίνουμε στην σελίδα https://nodejs.org/en/ και πατάμε το πράσινο κουμπί με την ονομασία <<Recommended For Most Users >>



Εικόνα 17 Εγκατάσταση NodeJS

Βήμα 2: Στην συνέχεια θα κατεβεί το installer , οπότε απλά πρέπει να το πατήσουμε για να ξεκινήσει η εγκατάσταση και πατάμε το next

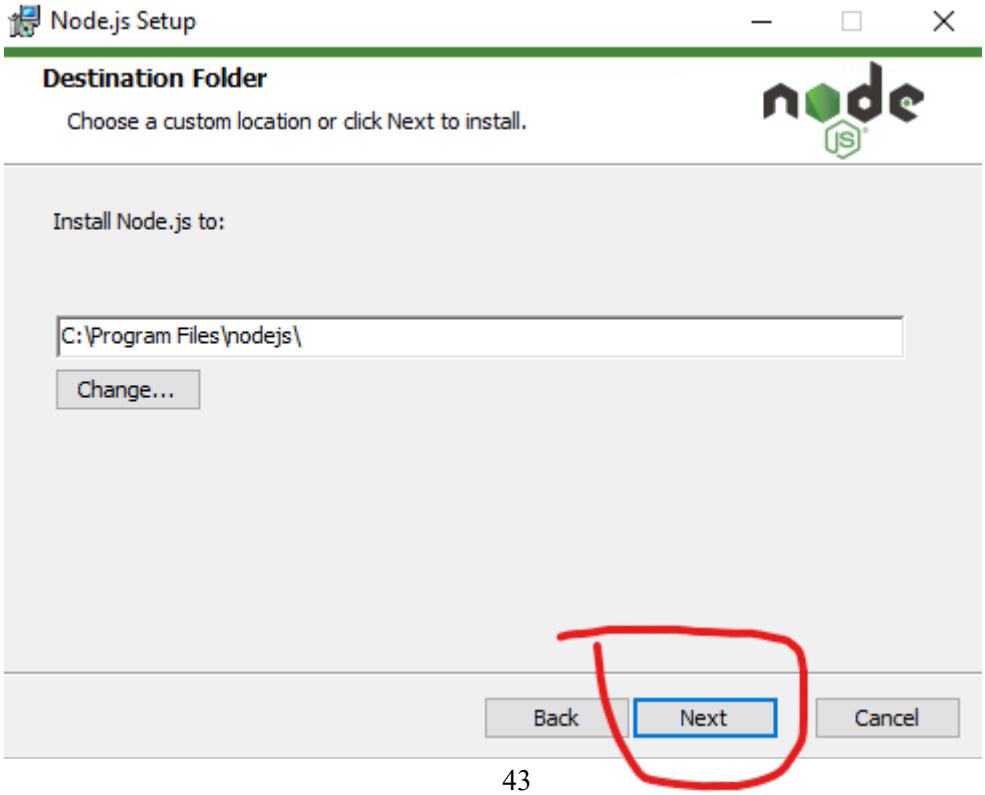


Εικόνα 18 Εγκατάσταση NodeJS

Βήμα 3: Στην συνέχεια κάνουμε αποδοχή στους όρους χρήσης και πατάμε next

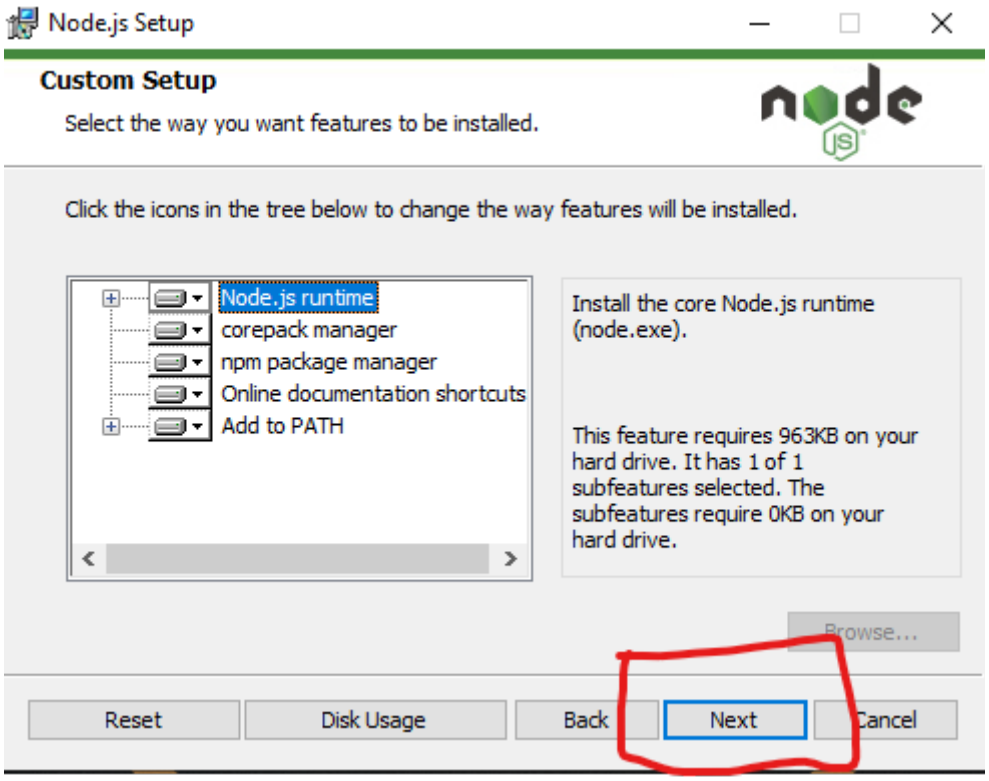
Εικόνα 19 Εγκατάσταση NodeJS

Βήμα 4: Στην συνέχεια επιλέγουμε την τοποθεσία που θέλουμε να γίνει η εγκατάσταση και πατάμε next



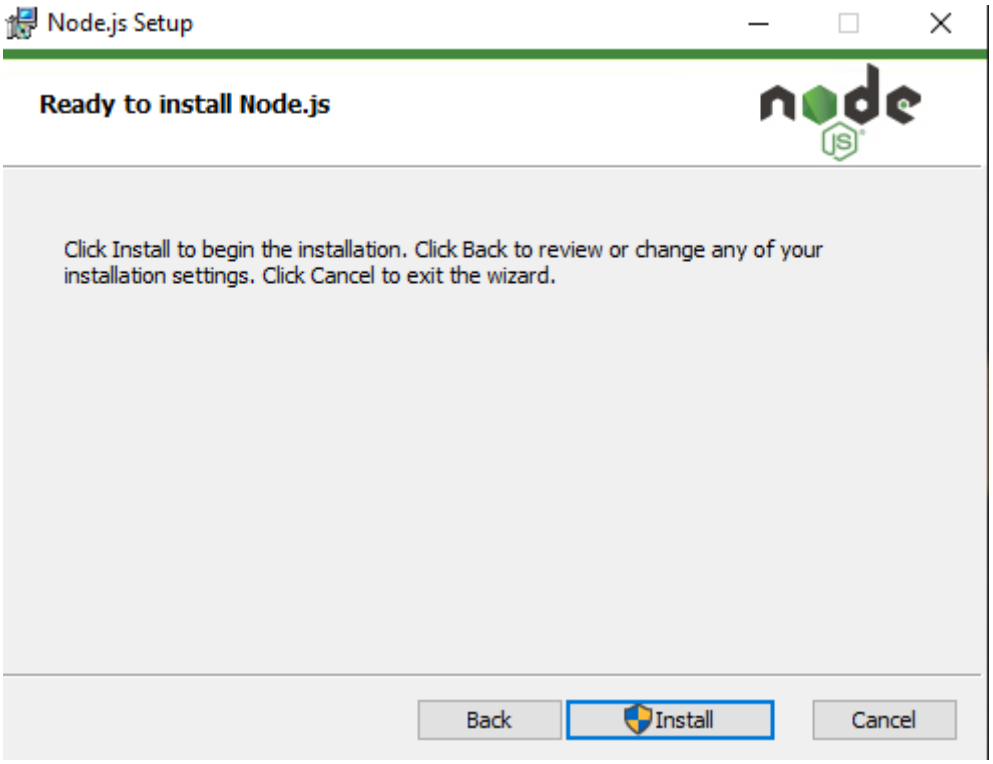
Εικόνα 20 Εγκατάσταση NodeJS

Βήμα 5: Στην συνέχεια αφήνουμε τα πάντα όπως είναι και πατάμε next



Εικόνα 21 Εγκατάσταση NodeJS

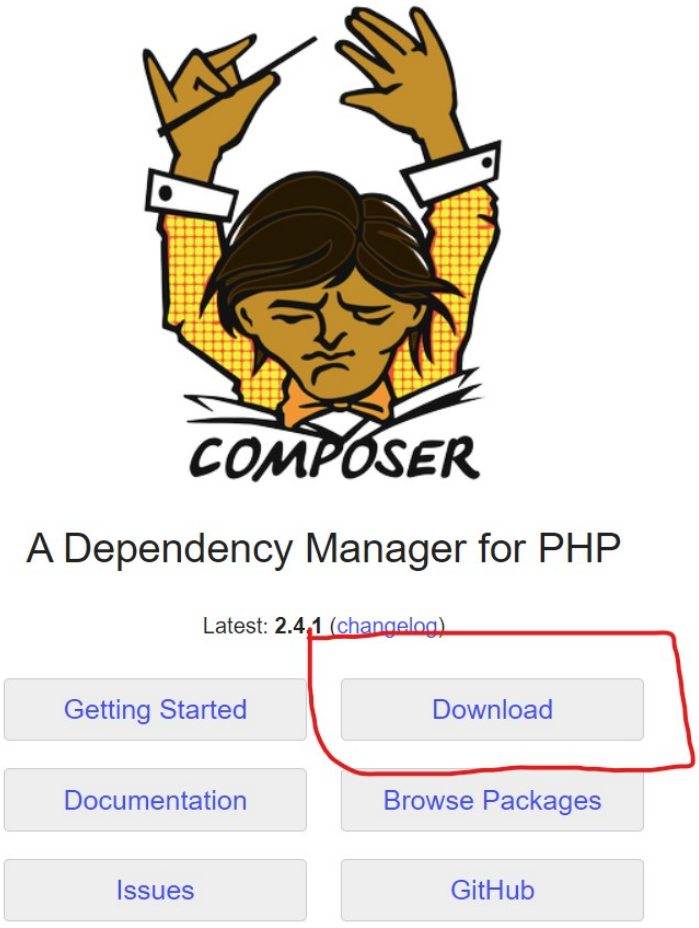
Βήμα 6: Στην συνέχεια πατάμε next και μετά install για να ξεκινήσει η εγκατάσταση.



Εικόνα 22 Εγκατάσταση NodeJS

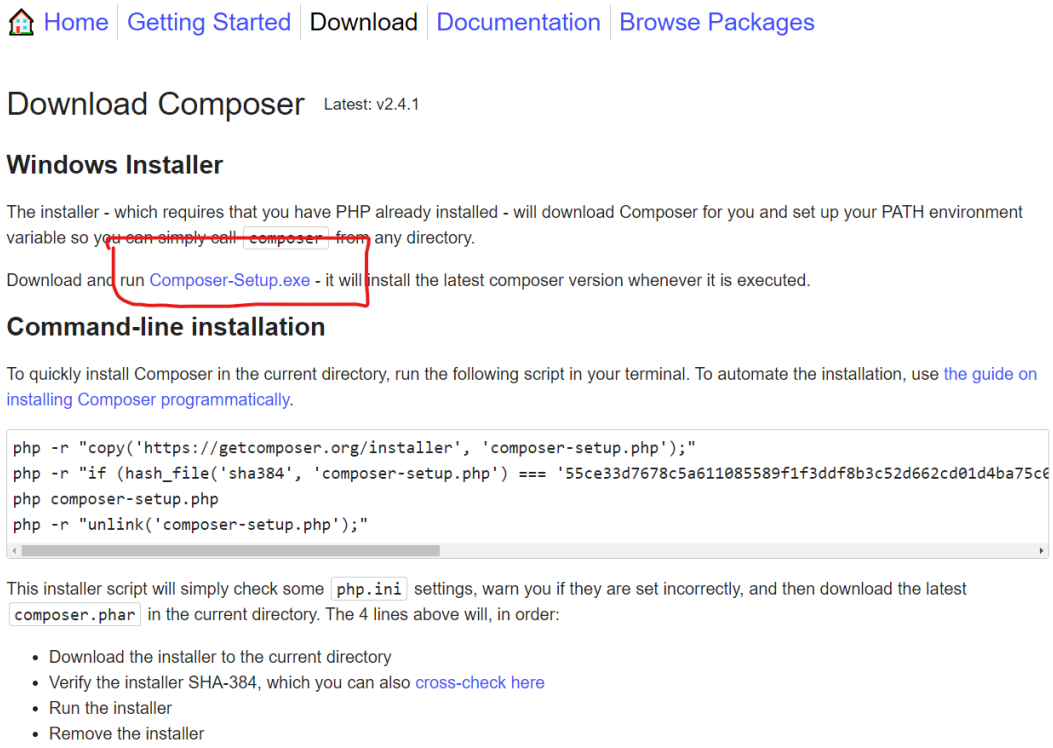
4.3 Εγκατάσταση Composer

Βήμα 1: Για την εγκατάσταση του composer πηγαίνουμε στην Σελίδα https://getcomposer.org/ και πατάμε το κουμπί που γράφει <<Download>>



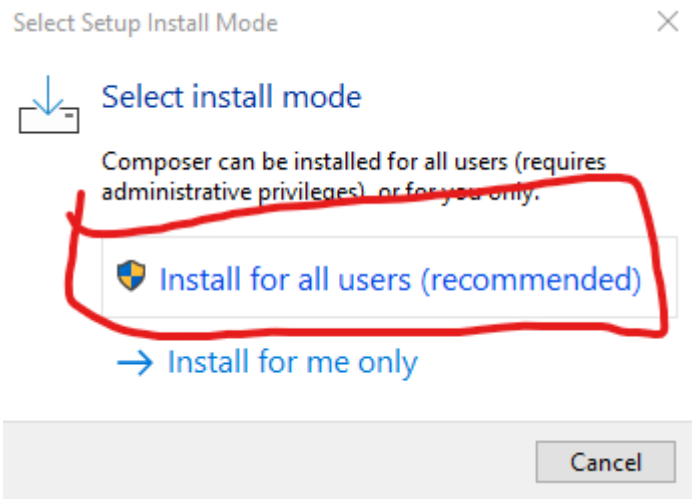
Εικόνα 23 Εγκατάσταση composer

Βήμα 2: Στην συνεχεία κατεβάζουμε το installer με την ονομασία <<Composer-Setup.exe>>



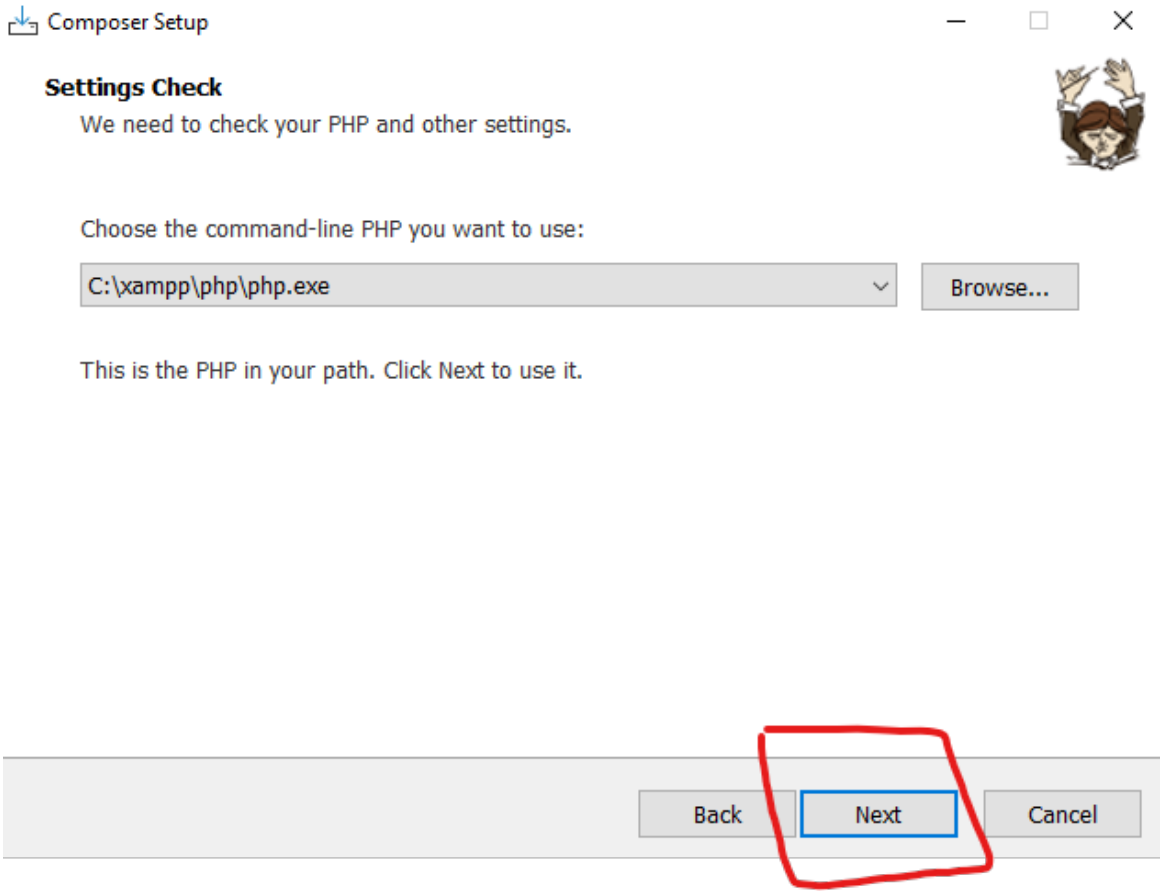
Εικόνα 24 Εγκατάσταση composer

Βήμα 3: Στην συνεχεία ανοίγουμε το installer και πατάμε <<install for all users>>



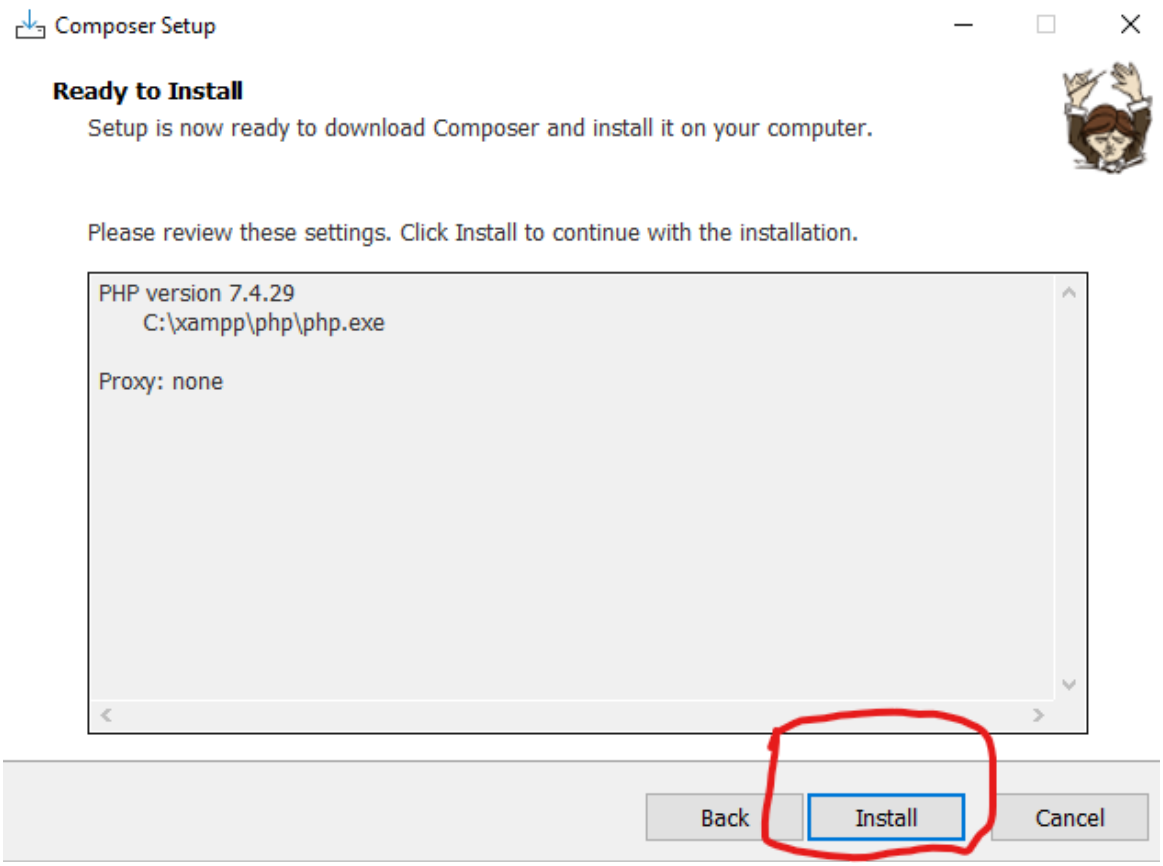
Εικόνα 25 Εγκατάσταση composer

Βήμα 4: Στην συνεχεία επιλέγουμε την τοποθεσία στην οποία είναι εγκατεστημένη η PHP στον υπολογιστή.



Εικόνα 26 Εγκατάσταση composer

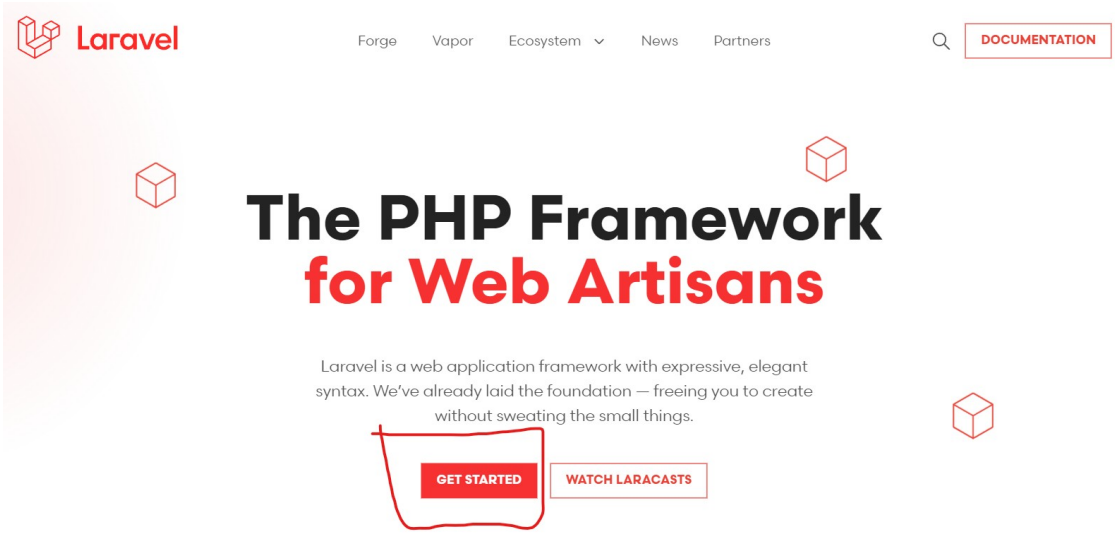
Βήμα 5: Στην συνεχεία πατάμε Finish για να τελειώσει η εγκατάσταση



Εικόνα 27 Εγκατάσταση composer

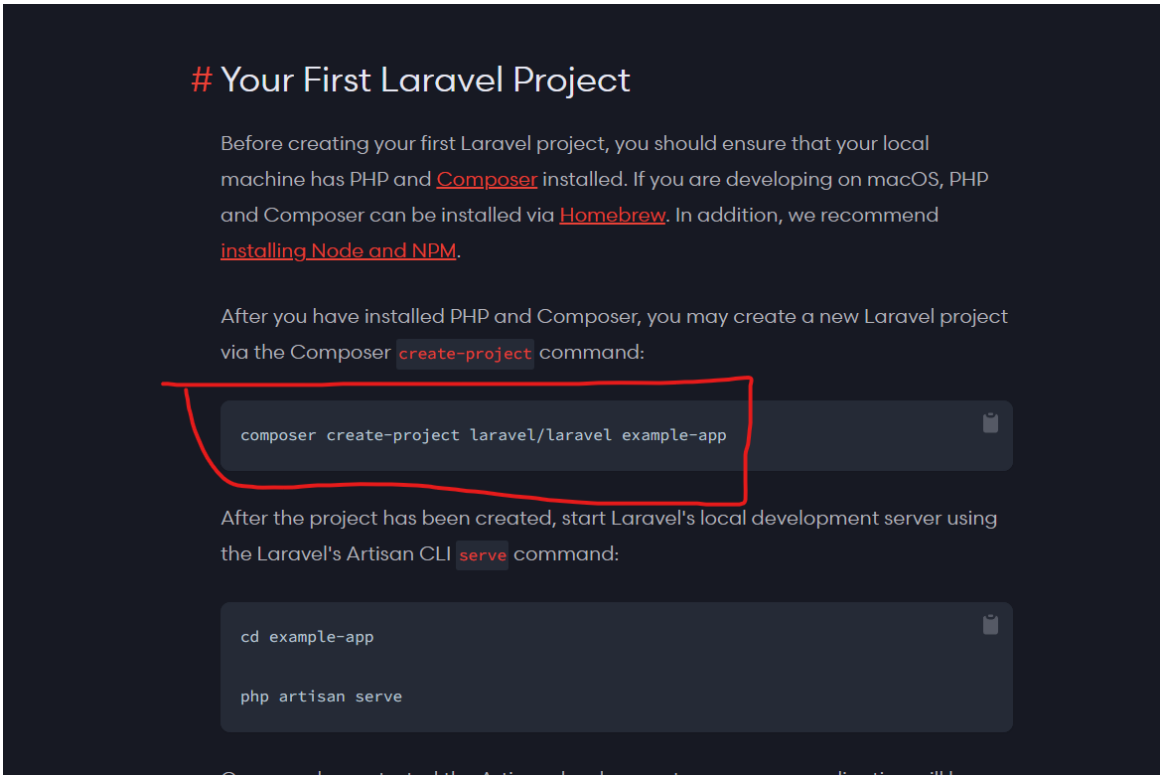
4.4 Αρχικοποίηση ενός Laravel Project

Βήμα 1: Πηγαίνουμε στην σελίδα https://laravel.com/ και πατάμε στο <<get Started>>



Εικόνα 28 Δημιουργία Laravel project

Βήμα 2: Πηγαίνουμε λίγο ποιο κάτω και κάνουμε αντιγραφή την εντολή για να ξεκινήσουμε ένα νέο Laravel project .

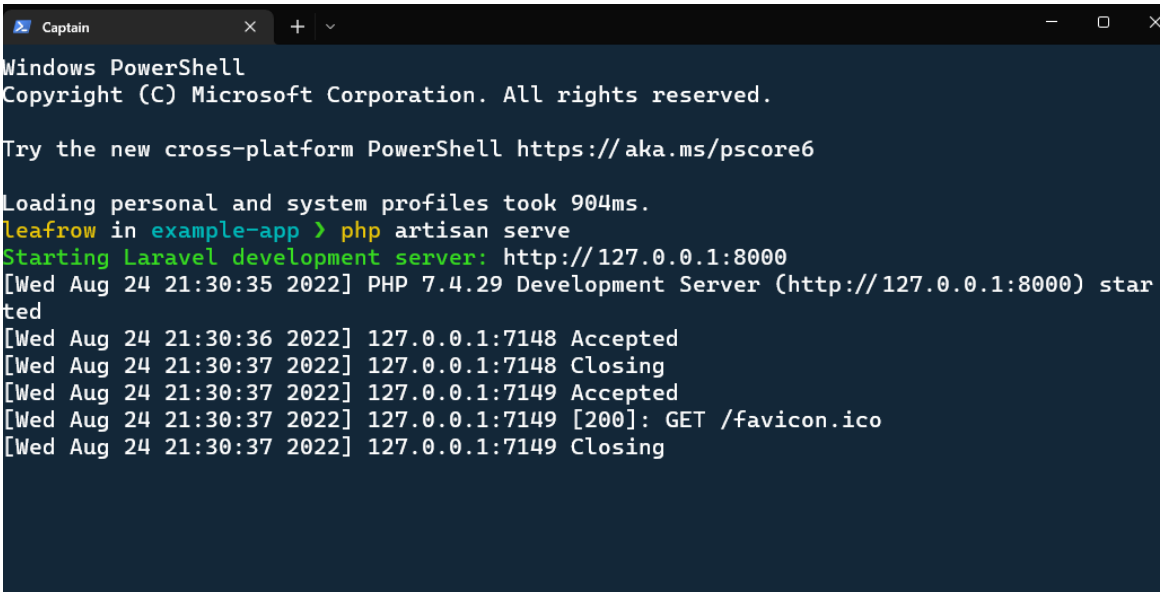


Εικόνα 29 Δημιουργία Laravel project

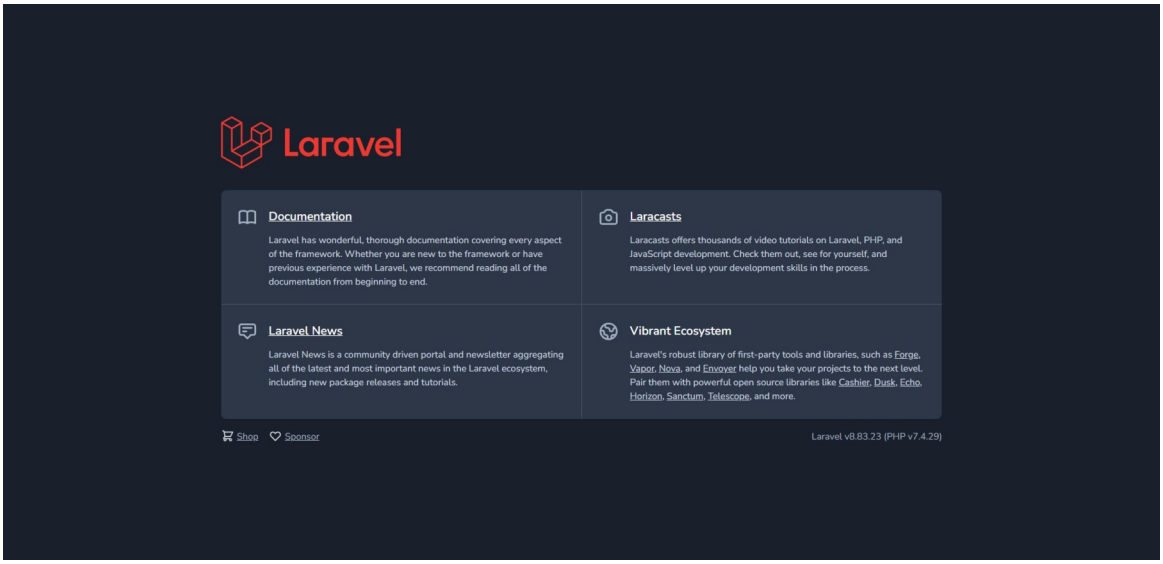
Βήμα 3: Στην συνεχεία ανοίγουμε ένα terminal και κάνουμε επικόλληση την εντολή και αν όλα πάνε καλά θα πρέπει να ξεκινήσει η δημιουργία του νέου Laravel Project.

Εικόνα 30 Δημιουργία Laravel project

Βήμα 4: Στην συνεχεία γράφουμε την εντολή php artisan serve και λογικά θα ανοίξει ο development server και θα αν πατήσουμε τον σύνδεσμο θα ανοίξει στην πόρτα 8000 σε ζωντανή μετάδοση η εφαρμογή.



Εικόνα 31 Δημιουργία Laravel project



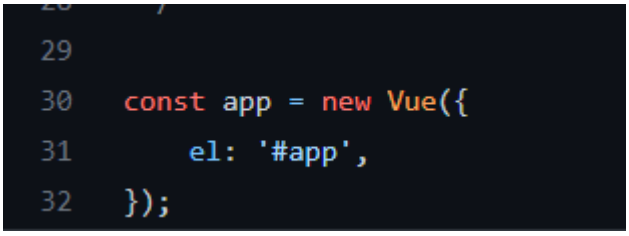
Εικόνα 32 Δημιουργία Laravel project

4.5 Εγκατάσταση VueJS σε Laravel project

Βήμα 1: Ανοίγουμε ένα τερματικό (command terminal) αν δεν το έχουμε ήδη ανοικτό από πριν και πληκτρολογούμε τις εντολές npm install vue και μετά npm install vue-template-compiler αυτές οι δυο εντολές θα εγκαταστήσουν την vue στο project μας. Αφού τρέξουν αυτές οι 2 εντολές τότε αν κοιτάξουμε στο αρχείο με όνομα package.json θα πρέπει να υπάρχουν στην κατηγορία devDependencies όπως φαίνεται και στην φωτογραφία παρακάτω. Τρέχοντας αυτές τις εντολές κάνουμε χρήση του NodeJS που κάναμε εγκατάσταση σε προηγούμενη ενότητα , το npm είναι ο node package manager που χρησιμοποιεί η nodeJS για την διαχείριση των πακέτων είτε για προσθήκη είτε για αφαίρεση αυτόν.

Εικόνα 33 εγκατάσταση vue

Βήμα 2: Στην συνεχεία ανοίγουμε το αρχείο app.js στο φάκελο resources → js → app.js και προσθέτουμε το κομμάτι κωδικά που φαίνεται στην φωτογραφία , με αυτόν τον τρόπο δημιουργούμε ένα καινούργιο αντικείμενο το οποίο θα κοιτάει στο id = #app



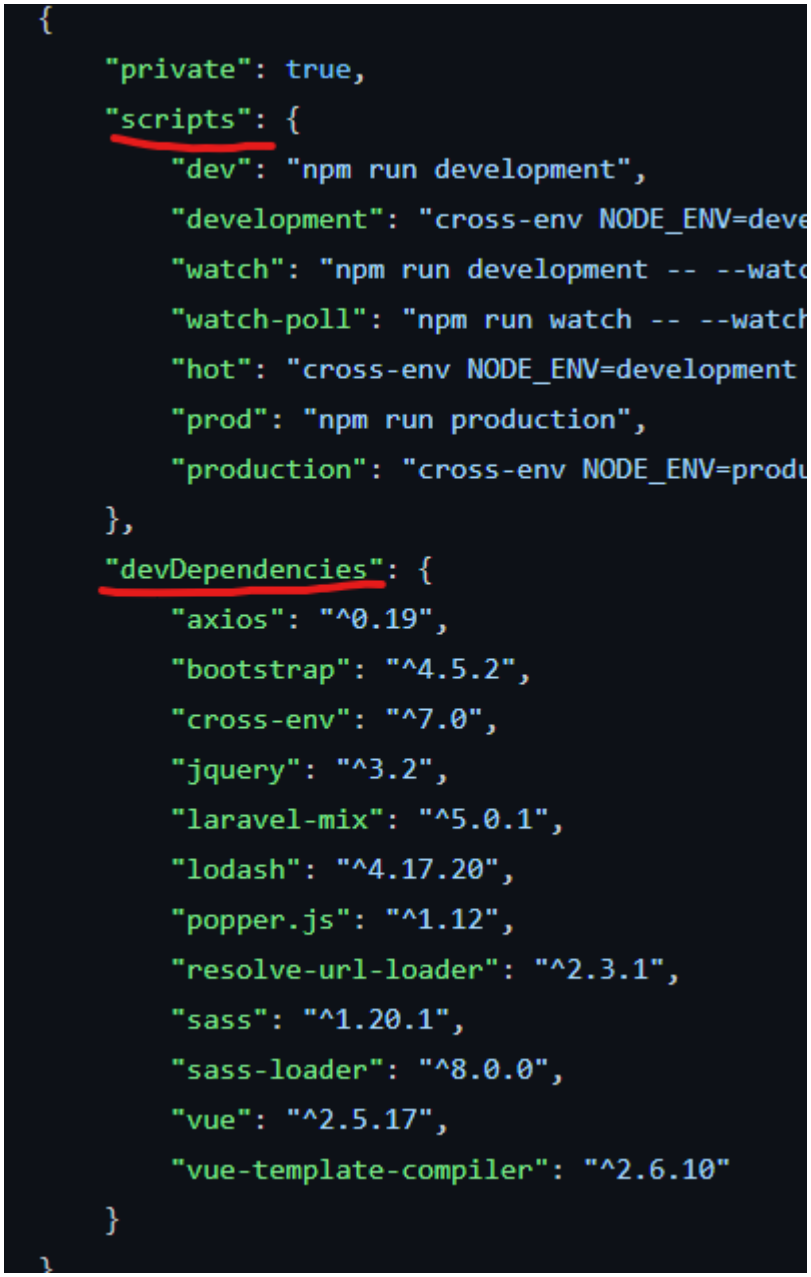
Εικόνα 34 εγκατάσταση vue

Βήμα 3: Στην συνεχεία δημιουργούμε το αρχείο app.blade.php μέσα στο φάκελο resources → views → layouts και συμπληρώνουμε αυτές τις εντολές που φαίνονται στην φωτογραφία παρακάτω. Με αυτόν τον τρόπο λέμε ότι τα components που έχουν φτιαχτεί θα πηγαίνουν στο αυτό το id = #app που συμπληρώσαμε ποιο πάνω. Το ίδιο ισχύει και για το script HTML tag το οποίο κοιτάει στο αρχείο που συμπληρώσαμε στο προηγούμενο βήμα. Τέλος το style HTML tag είναι για την CSS που θα χρειαστεί να γράψουμε.

```
11
12      <!-- Scripts -->
13      <script src="{{ asset('js/app.js') }}" defer></script>
14
15      <!-- Fonts -->
16      <link rel="dns-prefetch" href="//fonts.gstatic.com">
17      <link href="https://fonts.googleapis.com/css?family=Nunito" rel="stylesheet">
18
19      <!-- Styles -->
20      <link href="{{ asset('css/app.css') }}" rel="stylesheet">
21
22      <!-- Icons -->
23      <link href="{{ asset('themify/themify-icons.css') }}" rel="stylesheet">
24
25      <!-- Scripts -->
26      <script> window.csrf = "{{ csrf_token() }}"; </script>
27  </head>
28  <body>
29      <div id="app">
30          @include('includes.navbar')
31
32          <main class="py-4">
33              @yield('content')
34          </main>
35      </div>
36  </body>
37  </html>
```

Εικόνα 35 εγκατάσταση vue

Βήμα 4: Τέλος αφού τα κάνουμε όλα αυτά για να βεβαιωθούμε ότι όλα δουλεύουν όπως πρέπει ανοίγουμε ένα τερματικό (command terminal) και πληκτρολογούμε την εντολή npm install και μετά npm run dev. Η 1η εντολή θα κάνει εγκατάσταση όλες τις απαραίτητες βιβλιοθήκες που υπάρχουν στο package.json που φαίνεται και στην παρακάτω φωτογραφία , και η 2η εντολή θα καλέσει το Laravel-mix που είναι υπεύθυνο να ομαδοποιήσει τον κώδικα που αποτελείται από HTML, CSS, JS και να παράγει το εκτελέσιμο, στην συνεχεία αν όλα δουλέψουν σωστά με την εντολή php artisan serve ανοίγουμε τον development server και βλέπουμε αν τρέχει η εφαρμογή στην πόρτα 8000 , αν συμβεί αυτό τότε έχουμε εκτελέσει όλα τα βήματα με επιτυχία και είμαστε έτοιμη να συνεχίσουμε το χτίσιμο της εφαρμογής.

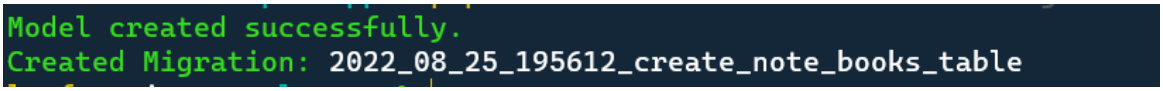


Εικόνα 36 αρχείο package.json

4.6 Δημιουργία πινάκων note_books & notes

Βήμα 1: Για ακόμη μια φορά θα χρησιμοποιήσουμε το πολύ δυνατό command line tool που μας προσφέρει η Laravel. Το 1ο μοντέλο που θα φτιάξει στην βάση δεδομένων είναι ο πινάκας NoteBook. Έχοντας ανοικτό ένα τερματικό (terminal) πληκτρολογήσουμε την εντολή php artisan make:model NoteBook –migration,Το πρώτο μέρος της εντολής θα δημιουργήσει το μοντέλο με όνομα NoteBook και το 2ο μέρος της εντολής (--migration) θα δημιουργήσει το αρχείο που μέσα σε αυτό θα προσθέσουμε τα πεδία της βάσης δεδομένων μας.

Εικόνα 37 εντολή δημιουργίας μοντέλου NoteBook



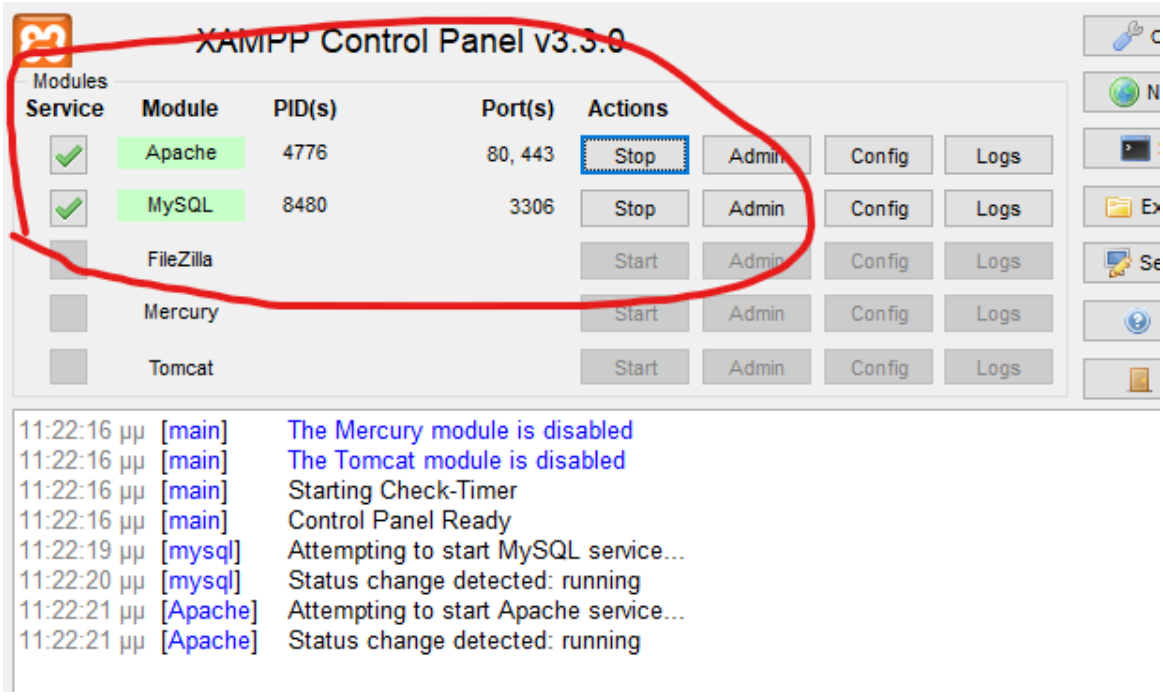
Εικόνα 38 μήνυμα στην κονσόλα

Βήμα 2: Ίδια διαδικασία και για το μοντέλο Notes με την εντολή php artisan make:model Note –migration και το αντίστοιχο μήνυμα στην κονσόλα.

Εικόνα 39 εντολή δημιουργίας μοντέλου Notes

Εικόνα 40 μήνυμα στην κονσόλα

Βήμα 3: Στην συνέχεια ανοίγουμε την εφαρμογή XAMPP και θέτουμε σε λειτουργιά τα πρώτα 2 πεδία δηλαδή την βάση δεδομένων MySQL και τον Apache, όπως φαίνεται και στην εικόνα κάτω η MySQL τρέχει στην πόρτα 3306 ο Apache στην πόρτα 80



Εικόνα 41 Apache control panel

Βήμα 4: Στην συνέχεια πατάμε το κουμπί Admin στην γραμμή που είναι η MySQL και μας κατευθύνει στο γραφικό περιβάλλον για την διαχείριση της βάσης δεδομένων το phpmyadmin. Εκεί πάνω δεξιά πατάμε το <<New>> και δημιουργούμε μια άδεια βάση δεδομένων με το όνομα notedb.

Βήμα 5: Στην συνέχεια ανοίγουμε το αρχείο create_note_books_table.php στον φάκελο database → migrations που είχε δημιουργηθεί με την εντολή που είχαμε τρέξει σε προηγούμενο βήμα και προσθέτουμε τα πεδία που θέλουμε να έχει αυτός ο πίνακας. Τα πεδία που θα έχει αυτός ο πίνακας είναι ένα id , ο τίτλος , το id του χρήστη στον οποίο ανήκει , ημερομηνίες και το ξένο κλειδί το ποιο αναφέρετε στον πίνακα users ώστε να γίνει η σύνδεση ένα προς πολλά.

```
$table->id();
$table->unsignedBigInteger('user_id');
$table->string('title');
$table->timestamps();

$table->foreign('user_id')->references('id')->on('users')->onDelete('cascade');
```

Εικόνα 42 note_books πεδία

Βήμα 6: Στην συνέχεια ανοίγουμε το αρχείο User.php , το αρχείο αυτό είναι το μοντέλο του χρήστη στην βάση δεδομένων οπότε πρέπει να συμπληρωθεί η συσχέτιση με τον πίνακα note_books όπως φαίνεται και στην φωτογραφία ότι ο χρήστης έχει πολλά note_books.

```php
public function notebooks()
{
    return $this->hasMany(NoteBook::class);
}
```

Εικόνα 43 σχέση User και note_books

Βήμα 7: Στην συνέχεια ανοίγουμε το αρχείο create_notes_table.php στον φάκελο database → migrations που είχε δημιουργηθεί με την εντολή που είχαμε τρέξει σε προηγούμενο βήμα και προσθέτουμε τα πεδία που θέλουμε να έχει αυτός ο πίνακας. Τα πεδία που θα έχει αυτός ο πίνακας είναι ένα id , το id του χρήστη στον οποίο ανήκει , ο τίτλος , το περιεχόμενο , ημερομηνίες και το ξένο κλειδί το ποιο αναφέρετε στον πίνακα note_books ώστε να γίνει η σύνδεση ένα προς πολλά.

```php
        $table->id();
        $table->unsignedBigInteger('note_book_id');
        $table->string('title');
        $table->text('content')->nullable();
        $table->timestamps();

        $table->foreign('note_book_id')->references('id')->on('note_books')->onDelete('cascade');
    });
```

Εικόνα 44 notes πεδία

Βήμα 8: Στην συνέχεια ανοίγουμε το αρχείο NoteBooks.php , το αρχείο αυτό είναι το μοντέλο του σημειωματάριου στην βάση δεδομένων οπότε πρέπει να συμπληρωθεί η συσχέτιση με τον πίνακα notes όπως φαίνεται και στην φωτογραφία ότι το σημειωματάριο έχει πολλές σημειώσεις και αντίστοιχα ότι ανήκει σε ένα χρήστη.

```php
public function user()
{
    return $this->belongsTo(User::class);
}

public function notes()
{
    return $this->hasMany(Note::class);
}
```

Εικόνα 45 σχέση notes και note_books

Βήμα 9: Στην συνέχεια ανοίγουμε το αρχείο .env ψάχνουμε την γραμμή με το όνομα DB_DATABASE και αντικαθιστούμε την τιμή που έχει με το όνομα που δώσαμε πριν στην βάση δεδομένων που φτιάξαμε δηλαδή το notedb σε ένα τερματικό (termimal) πληκτρολογούμε την εντολή php artisan migrate αυτή η εντολή θα μετατρέψει το μοντέλο που έχουμε φτιάξει σε πίνακα στην βάση δεδομένων μας. Αν τρέξει η εντολή θα εμφανιστεί στην κονσόλα το μήνυμα ότι έχουν δημιουργηθεί οι πίνακες στην βάση δεδομένων . Οπότε αν ανοίξουμε την βάση δεδομένων θα δούμε ότι πλέον δεν είναι άδεια αλλά έχει μερικούς πίνακες που μεταξύ αυτόν είναι και οι 3 πίνακες που μας ενδιαφέρουν δηλαδή ο πίνακας users ο πίνακας note_books και ο πίνακας notes.

```
[redacted] in example-app ❯ php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table (59.18ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table (34.57ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated:  2019_08_19_000000_create_failed_jobs_table (37.71ms)
Migrating: 2019_12_14_000001_create_personal_access_tokens_table
Migrated:  2019_12_14_000001_create_personal_access_tokens_table (42.99ms)
Migrating: 2022_08_25_195612_create_note_books_table
Migrated:  2022_08_25_195612_create_note_books_table (23.65ms)
Migrating: 2022_08_25_195835_create_notes_table
Migrated:  2022_08_25_195835_create_notes_table (22.60ms)
```

Εικόνα 46 table migrations

Table	Action	Rows	Type	Collation	Size	Overhead
failed_jobs	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
migrations	Browse Structure Search Insert Empty Drop	6	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
notes	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
note_books	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
password_resets	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
personal_access_tokens	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	48.0 KiB	-
users	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-

Εικόνα 47 πίνακες στην βάση δεδομένων

4.7 Δημιουργία NoteBook Controller

Εφόσον έχουμε φτιάξει τα μοντέλα της βάσης (NoteBook) με τα πεδία και τις συσχετίσεις μπορούμε να τα χρησιμοποιήσουμε πλέον. Οπότε το 1ο πράγμα είναι να φτιαχτεί ο controller. Ανοίγουμε ένα τερματικό (command terminal) και πληκτρολογούμε την εντολή php artisan make:controller NoteBookController αυτή η εντολή θα φτιάξει μέσα στον φάκελο app → Http → Controllers ένα αρχείο με όνομα NoteBookController.php οπότε εκεί μέσα θα γραφτεί ο κώδικας που αφορά την λογική πίσω από το μοντέλο NoteBook. Η κύρια δουλειά αυτού του Controller είναι να χειρίζεται τα αιτήματα που θα έρχονται από το user interface της εφαρμογής. Αυτά τα αιτήματα αποτελούνται από το αίτημα GET το οποίο είναι όταν ο χρήστης θέλει να πάρει κάποια δεδομένα από την βάση δεδομένων και να τα δει, το αίτημα POST το οποίο είναι όταν ο χρήστης θέλει να δημιουργήσει ένα καινούργιο NoteBook αντικείμενο στην βάση δεδομένων, το αίτημα EDIT το οποίο χρησιμοποιείτε όταν ο χρήστης θέλει να τροποποίηση ένα NoteBook αντικείμενο της βάσεις δεδομένων και να αλλάξει τιμές σε κάποια η σε όλα τα πεδία και τέλος είναι το DELETE το οποίο χρησιμοποιείτε όταν ο χρήστης θέλει να διαγράψει ένα NoteBook αντικείμενο από την βάση δεδομένων.

4.8 Δημιουργία Note Controller

Εφόσον έχει φτιαχτεί το μοντέλο της βάσης (Note) με τα πεδία και τις συσχετίσεις μπορούμε να το χρησιμοποιήσουμε πλέον. Οπότε το 1ο πράγμα είναι να φτιαχτεί ο controller. Ανοίγουμε ένα τερματικό (command terminal) και πληκτρολογούμε την εντολή php artisan make:controller NoteController αυτή η εντολή θα φτιάξει μέσα στον φάκελο app → Http → Controllers ένα αρχείο με όνομα NoteController.php οπότε εκεί μέσα θα γραφτεί ο κώδικας που αφορά την λογική πίσω από το μοντέλο Note. Η κύρια δουλειά αυτού του Controller είναι να χειρίζεται τα αιτήματα που θα έρχονται από το user interface της εφαρμογής. Αυτά τα αιτήματα αποτελούνται από το αίτημα GET το οποίο είναι όταν ο χρήστης θέλει να πάρει κάποια δεδομένα από την βάση δεδομένων και να τα δει, το αίτημα POST το οποίο είναι όταν ο χρήστης θέλει να δημιουργήσει ένα καινούργιο Note αντικείμενο στην βάση δεδομένων, το αίτημα EDIT το οποίο χρησιμοποιείτε όταν ο χρήστης θέλει να τροποποίηση ένα Note αντικείμενο της βάσεις δεδομένων και να αλλάξει τιμές σε κάποια η σε όλα τα πεδία και τέλος είναι το DELETE το οποίο χρησιμοποιείτε όταν ο χρήστης θέλει να διαγράψει ένα Note αντικείμενο από την βάση δεδομένων. Η κυρία διαφορά με τον controller NoteBook είναι ότι για να μπορεί χρήστης να δημιουργήσει ένα αντικείμενο Note θα πρέπει πρώτα να έχει δημιουργήσει ένα αντικείμενο NoteBook. Με απλά λόγια δεν μπορεί να υπάρξει ένα Note αντικείμενο χωρίς να υπάρχει τουλάχιστον ένα αντικείμενο NoteBook στο οποίο να ανήκει ένα Note το οποίο είναι κομμάτι της σχέσης μεταξύ των δυο μοντέλων.

4.9 Δημιουργία Authentication Middleware

- **Τι είναι το Middleware**

Το Laravel Middleware είναι μια προγραμματιστική γέφυρα μεταξύ ενός αιτήματος (request) και μιας αντίδρασης (response). Η Laravel ενσωματώνει ένα ενδιάμεσο λογισμικό (middleware) που επιβεβαιώνει εάν ο χρήστης της εφαρμογής έχει επαληθευτεί ή όχι. Εάν ο πελάτης επιβεβαιωθεί, κατευθύνετε στην αρχική σελίδα διαφορετικά, κατευθύνετε στη σελίδα σύνδεσης η γενικά σε κάποια σελίδα που ο προγραμματιστής έχει επιλέξει να κατευθύνει τον χρήστη σε περίπτωση που δεν ικανοποιηθεί μια συνθήκη.

- **Γιατί να χρησιμοποιήσω Middleware**

Με ποιο απλά λόγια το middleware είναι ένα επαναχρησιμοποιήσιμο κομμάτι κώδικα το οποίο χρησιμοποιείτε για μια συγκεκριμένη ενέργεια αρκετά συχνά οπότε διευκολύνει τον προγραμματιστή να γραφτεί μια φορά και να χρησιμοποιείται κάθε φορά όπου χρειαστεί. Αυτό κάνει τον κώδικα της εφαρμογής ποιο ευανάγνωστο πιο συντηρήσιμο και πιο εύκολα επεκτάσιμο.

- **Δημιουργία του Middleware**

Για να δημιουργήσουμε το middleware που χρειαζόμαστε σε αυτήν την εφαρμογή ανοίγουμε ένα τερματικό (command terminal) και πληκτρολογούμε την εντολή php artisan make:middleware Authenticate με αυτήν την εντολή θα δημιουργηθεί μέσα στο φάκελο app → Http → Middleware ένα αρχείο με όνομα Authenticate.php και μέσα θα γράψουμε το κομμάτι κώδικα το οποίο θα χρησιμοποιούμε κάθε φορά που χρειάζεται να δούμε αν στην ενέργεια που κάνει ο χρήστης είναι η όχι εξουσιοδοτημένος για αυτήν την ενέργεια. Ένα απλό παράδειγμα είναι όταν ο χρήστης θέλει να δημιουργήσει ένα αντικείμενο αλλά πριν το κάνει αυτό πρέπει πρώτα να κάνει είσοδο στον λογαριασμό του με το email και το password του. Αν δεν κάνει κάτι τέτοιο τότε δεν μπορεί να εισέλθει στην εφαρμογή και να την χρησιμοποιήσει. Αυτό από την πλευρά του χρήστη , όσον αφορά την πλευρά του κώδικα όταν θα έρθει το request από τον χρήστη στον server τότε ο controller που είναι υπεύθυνος για το login θα δει αυτό το αίτημα και θα καλέσει το συγκεκριμένο middleware παίρνοντάς τα στοιχειά του χρήστη . Το middleware θα ψάξει στην βάση δεδομένων και αν βρει τον συγκεκριμένο χρήστη τότε θα επιστρέψει ένα οκ στον controller και ο controller θα επιστρέψει με την σειρά του το τελικό οκ ότι έχει γίνει επαλήθευση και οτι μπορεί να εισέλθει στην εφαρμογή και να αρχίσει να την χρησιμοποιεί. Από την άλλη πλευρά αν το middleware δεν βρει τον Χρήστη στην βάση δεδομένων σημαίνει ότι δεν υπάρχει χρήστης με αυτά τα συγκεκριμένα στοιχειά οπότε και θα επιστρέψει ότι δεν είναι εξουσιοδοτημένος, και ο controller με την σειρά θα επιστρέψει ότι αυτός ο χρήστης με τα συγκεκριμένα στοιχειά δεν υπάρχει όποτε θα

πρέπει να δημιουργηθεί ένας λογαριασμός. Παρακάτω εμφανίζεται ένα κομμάτι κωδικά από το συγκεκριμένο middleware.

```
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @param  string|null  $guard
     * @return mixed
     */
    public function handle($request, Closure $next, $guard = null)
    {
        if (Auth::guard($guard)->check()) {
            return redirect(RouteServiceProvider::HOME);
        }

        return $next($request);
    }
}
```

Εικόνα 48 authentication Middleware

4.10 Δημιουργία Δρομολογητών (Routers)

Οι δρομολογητές στο Laravel έχουν την δική τους θέση μέσα στους φακέλους της εφαρμογής και ουσιαστικά είναι το URL στο οποίο θα σταλθούν τα αιτήματα (requests) του χρήστη όσο χρησιμοποιεί την εφαρμογή. Όταν ο χρήστης θέλει να δημιουργήσει ένα αντικείμενο Note τότε όταν θα πατήσει το κουμπί create ο κώδικας θα κάνει ένα αίτημα (request) προς τον διακομιστή (server) ζητώντας να δημιουργήσει αυτό το αντικείμενο και να το αποθηκεύσει στην βάση δεδομένων. Το URL στο οποίο θα στείλει το αίτημα request είναι αυτό που λέμε route. Εκτός όμως από αυτήν την δουλειά οι δρομολογητές (routers) είναι υπεύθυνοι και για μια ακόμα πολύ σημαντική λειτουργιά της εφαρμογής η οποία είναι να αναθέσουν στους controllers για το σε ποια σημεία θα καλεστούν ώστε να εκτελεστεί το κομμάτι κώδικα που έχουν μέσα τους αυτές οι συναρτήσεις. Με ποιο απλά λόγια οι δρομολογητές (routers) αναθέτουν στους controllers για το ποια routes είναι υπεύθυνα και σε ποιο σημείο θα καλεστούν και να εκτελέσουν τον κώδικα του περιέχουν για αυτό το συγκεκριμένο route.

```
Route::resource('note', 'NoteController')->only(['store', 'update', 'destroy']);
```

Εικόνα 49 Παράδειγμα Δρομολογητή (Router)

Η παραπάνω φωτογραφία είναι ένα παράδειγμα κώδικα ενός Δρομολογητή (Router) στον φάκελο app → routes → web.php αρχείο του project. Το πρώτο σκέλος του κώδικα αναφέρεται σε 2 πράγματα. Το πρώτο είναι το URL στο οποίο φιλοξενείται αυτή η συγκεκριμένη λογική του κώδικα το οποίο είναι το <<note>>. Το <<note>> αντικατοπτρίζει το κομμάτι του URL το οποίο κάθε φορά αλλάζει , για παράδειγμα εδώ έχουμε το βασικό URL (base URL) το οποίο είναι το http://127.0.0.1:8000 αν πάρουμε αυτό σαν δεδομένο τότε το <<note>> σημαίνει ότι το URL το οποίο κοιτάει αυτός ο δρομολογητής είναι το http://127.0.0.1:8000/note . Οπότε κάτω από την στέγη αυτή κοιτάει αυτός ο router. Το 2ο μέρος του πρώτου σκέλους είναι το σε ποιο controller αναφέρεται η αλλιώς ποιος controller είναι υπεύθυνος για αυτόν τον δρομολογητή (router) . Στην συγκεκριμένη περίπτωση είναι ο controller με την ονομασία NoteController ο οποίος έχει αναφερθεί και εξηγηθεί ποιο πάνω. Το 2ο σκελος του κώδικα αυτού αναφέρει 3 πράγματα τα <<store>> , <<update>> , <<destroy>> τα οποία είναι τα ονόματα μερικών από των μεθόδων της κλάσης NoteController. Οπότε εδώ αυτό που συμβαίνει είναι ότι σε αυτόν τον δρομολογητή θα χρησιμοποιηθούν αυτές οι τρεις μέθοδοι μόνο από την κλάση NoteController.

```
Route::post('upload', 'NoteBookController@upload');
```

Εικόνα 50 Παράδειγμα Δρομολογητή (Router) 2

Ένα ακόμα παράδειγμα κώδικα είναι αυτό στην παραπάνω φωτογραφίας όπου εδώ έχουμε μια παρόμοια λογική με το βασικό URL (Base URL) να είναι το http://127.0.0.1:8000 οπότε στο πρώτο σκέλος βλέπουμε να αναγράφετε το <<upload>> άρα έχουμε http://127.0.0.1:8000/upload αυτό είναι το λειτουργικό και τελικό URL για αυτού του είδους την ενέργεια η οποία είναι το ανέβασμα αρχείων στη βάση δεδομένων όπως κείμενο φωτογραφίες βίντεο κτλ. Στο δεύτερο σκέλος έχουμε τον Controller που είναι υπεύθυνος για να αναλάβει αυτήν την ενέργεια και τώρα με μια διαφορετική σύνταξη δηλώνεται και η μέθοδος της κλάσης NoteController . Οπότε εδώ βλέπουμε ότι η μέθοδος <<upload>> της κλάσης NoteController θα κάνει την δουλειά που χρειάζεται για αυτό το κομμάτι κώδικα.

4.11 Database Seeding

Στην ενότητα αυτή θα εξηγηθεί το Database Seeding στην Laravel το οποίο με ποιο απλά λόγια είναι το γέμισμα της βάσης δεδομένων με ψεύτικες και δοκιμαστικές τιμές για να δοκιμάσουμε την λειτουργικότητα της εφαρμογής χωρίς να χρειαστεί να κάνουμε χειροκίνητα γεμίσματα στην βάση δεδομένων. Ένα πολύ απλό και καλό παράδειγμα είναι η δημιουργία χρηστών, σε ένα υποθετικό σενάριο θέλουμε να δημιουργήσουμε κάποιους χρήστες για να δουλέψουμε την εφαρμογή μας. Σε αυτήν την περίπτωση δεν θα κάτσουμε να δημιουργήσουμε αυτούς τους χρήστες έναν έναν διότι μπορεί να θέλουμε 200 η 300 η και 1000 η και παραπάνω οπότε το Laravel Framework μας παρέχει αυτό το εργαλείο τον Database Seeder ο οποίος αναλαμβάνει με απλό κώδικα να δημιουργήσει από την αντικείμενα από μια η πολλές κλάσεις ώστε να φτιάξει αυτά τα δεδομένα ώστε να υπάρχει ένας πίνακας γεμάτος με δεδομένα πίνακας ώστε να χρησιμοποιηθούν τα δεδομένα που περιέχει για δοκιμές στην εφαρμογή.

```
    public function run()
    {
        $notebooks = NoteBook::take(10)->get();

        foreach ($notebooks as $notebook) {
            $notes = factory(Note::class, 10)->make();

            $notebook->notes()->saveMany($notes);
        }
    }
}
```

Εικόνα 51 Database Seeding

Στην παραπάνω εικόνα βλέπουμε ένα παράδειγμα κώδικα για το πως να δημιουργήσουμε κάποια NoteBook αντικείμενα. Ουσιαστικά στην αρχή παίρνουμε 10 notebooks αντικείμενα από την βάση δεδομένων και τα αποθηκεύουμε σε μια μεταβλητή και στην συνέχεια με μια foreach για κάθε αντικείμενο notebook δημιουργούνται 10 αντικείμενα note. Έτσι στο τέλος καταλήγουν να υπάρχουν στην βάση δεδομένων 10 notebooks με 10 notes το κάθε notebook. Με έναν κώδικα 5 γραμμών. Σε περίπτωση που χρειαστούν παραπάνω δεδομένα με μια αλλαγή του νούμερου από 10 σε 100 θα δημιουργηθούν 100 αντικείμενα κτλ.

4.12 Eloquent ORM

Το Eloquent ORM του Laravel Framework με το ακρωνύμιο των λέξεων Object Relational Mapping και χρησιμοποιείται για να αλληλοεπιδρά με σχεσιακές Βάσεις Δεδομένων. Βοηθάει τον προγραμματιστή να μπορέσει να γράψει SQL γλωσσά προγραμματισμού αλλά με ποιο εύκολο και ασφαλή τρόπο. Έχει ένα σύνολο από έτοιμες μεθόδους που χρησιμοποιούνται για την δημιουργία την διαγραφή την ενημέρωση και την άντληση δεδομένων από την βάση δεδομένων.

```
// Eloquent ORM syntax
$notebooks = NoteBook::with('notebook.note')->get();

// SQL SYNTAX
SELECT * FROM noteBook JOIN note ON notebook.id = note.note_book_id;
```

Εικόνα 52 Database Seeding

Η παραπάνω φωτογραφία απεικονίζει την διαφορά μεταξύ της εντολής της PHP με το eloquent ORM και της αντίστοιχης εντολής SQL. Οι 2 αυτές εντολές κάνουν ακριβώς το ίδιο πράγμα απλά με το eloquent ORM είναι πιο εύκολο και πιο ασφαλής όταν πρόκειται για κώδικα PHP. Στην 1η περίπτωση του Eloquent ORM με την εντολή with από το μοντέλο NoteBook παίρνουμε όλα τα πεδία και το έξτρα πεδίο των notes που ανήκει σε άλλο πίνακα με αυτόν τον απλό τρόπο. Στην 2η περίπτωση έχουμε την κλασσική εντολή SQL όπου με την εντολή JOIN ενώνονται 2 πίνακες και παίρνουμε όλα τα πεδία από τον πίνακα NoteBook.

4.13 Laravel Resources

Το επόμενο βήμα εφόσον έχει φτιαχτεί το backend κομμάτι με την γλωσσά PHP είναι να φτιαχτεί και το κομμάτι της διεπαφής του χρήστη (user interface) . Το Laravel Resourses λοιπόν είναι το σημείο στο οποίο γράφεται ο κώδικας της εφαρμογής που αφορά το user interface και περιέχει αρχεία γραμμένα με HTML , CSS , JavaScript και το html template της Laravel το blade. Με το Blade Templating Engine έχουμε στη διάθεσή μας τις γνωστές Δομές Επιλογής (if/else/elseif/unless) και Δομές Επανάληψης (for/foreach/while). Τέλος, είναι πολύ σημαντικό στα αρχεία που χρησιμοποιούμε τη σύνταξη του Blade να έχουν την κατάληξη .blade.php

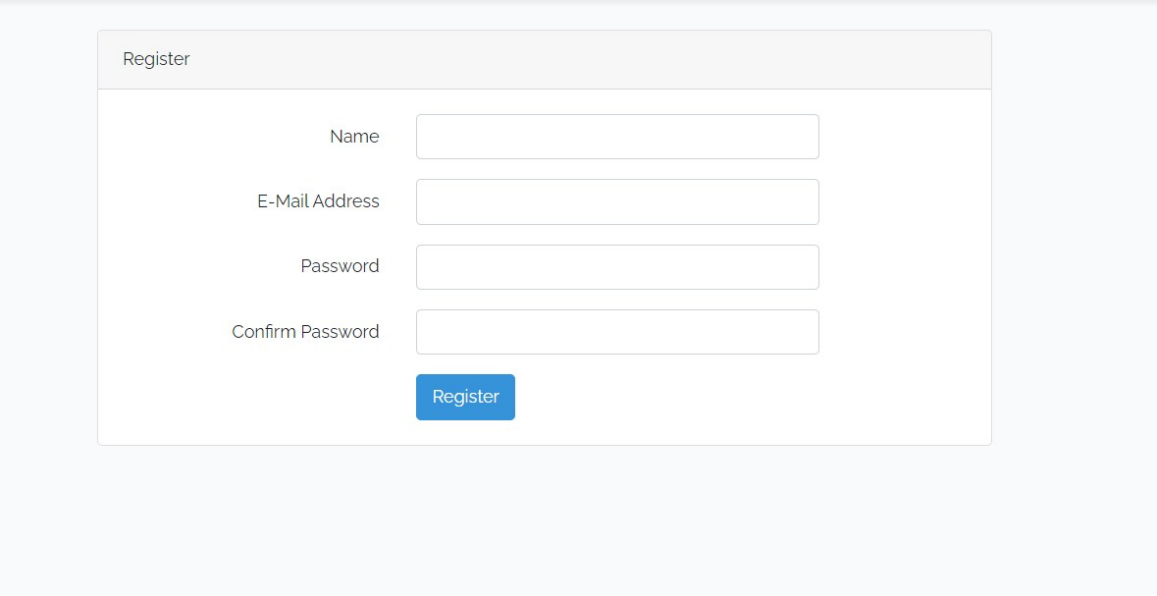
4.14 Laravel Views

Με τη χρήση του Blade Templating Engine έχουμε τη δυνατότητα να επεκτείνουμε Views από άλλα Views. Τα Views τα χωρίζουμε σε κάποιες κατηγορίες για να είναι πιο διαχειρίσιμα και επεκτάσιμα. Έχουμε τον φάκελο με τα Layouts ο όποιος έχει μέσα αρχεία τα οποία είναι κοινά σε όλα τα κομμάτια κώδικα και για να μην τα γράφουμε πολλές φορές υπάρχουν εκεί ώστε να τα χρησιμοποιήσουμε μια φορά και να υπάρχουν παντού. Στην συνεχεία έχουμε τον φάκελο με τα includes τα οποία είναι επεκτάσεις του κυρίως κώδικα τα οποία υπάρχουν για να κάνουν τον κώδικα μικρότερο σε επέκταση και πιο ευανάγνωστο και διαχειρίσιμο . Οπότε όταν μια λειτουργιά της εφαρμογής μπορεί να είναι 300 γραμμές κώδικα αυτό μπορεί να χωριστεί σε 3 ισομερή κομμάτια των 100 γραμμών ώστε να είναι ποιο εύκολο στην διαχείριση και στην επέκταση στο μέλλον. Παρακάτω θα αναλύσουμε κάποιες από τις βασικότερες «εντολές» που χρησιμοποιούμε στα Views.

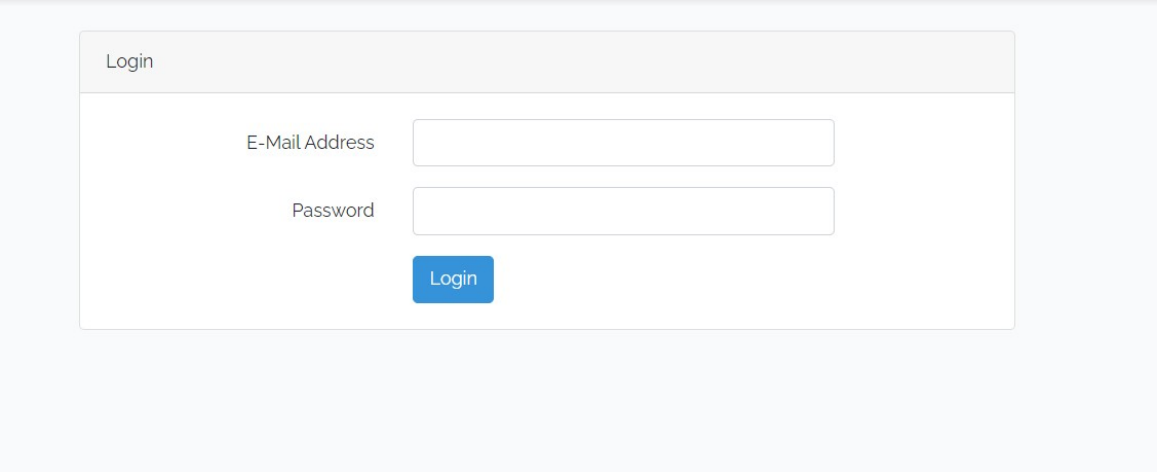
- **@section:** Τη χρησιμοποιούμε στο child view για να επεκτείνουμε το layout.
- **@yield:** Τη χρησιμοποιούμε στο parent view για να εμφανίσουμε τα αντίστοιχα περιεχόμενα του child view.
- **@extends:** Τη χρησιμοποιούμε στο child view διότι επεκτείνουμε το layout από το parent view. Μέσα στο @extends ορίζεται το layout που θα κληρονομήσει το child view.
- **@parent:** Τη χρησιμοποιούμε στο child view καθώς όταν ένα @section έχει οριστεί στο parent view και το ορίσουμε ξανά στο child view, το περιεχόμενο του **@section:** στο child view υπερκαλύπτει το περιεχόμενο του @section στο parent view.

4.15 Login & Register Pages

Οι 2 πρώτες σελίδες που συναντά κάποιος με το που ανοίξει την εφαρμογή. Το Login page είναι η σελίδα στην οποία με την προϋπόθεση ότι ο χρήστης έχει ήδη λογαριασμό, βάζει τα στοιχειά του εισέρχεται στην εφαρμογή και η Register page είναι η σελίδα στην οποία ο χρήστης δημιουργεί έναν λογαριασμό με τα στοιχειά τα οποίο βάζει και μετά εισέρχεται στην σελίδα. Αυτές οι 2 σελίδες έχουν κάποια κοινά σημεία στον κώδικα τα οποία φαίνονται και στις παρακάτω φωτογραφίες. Στις φωτογραφίες βλέπουμε ότι τα 2 από τα 4 πεδία είναι κοινά , το πεδίο για το email του χρήστη και το πεδίο για τον κωδικό πρόσβασης.



Εικόνα 53 Register Page



Εικόνα 54 Login Page

4.16 Vue Components

Στην συνέχεια αφού έχουν φτιαχτεί οι 2 σελίδες Login και Register δημιουργούμε έναν φάκελο app → resources → js → components όπου εκεί μέσα θα φτιαχτεί το component NoteBooks.vue το οποίο θα αποτελείται από 3 μέρη, το HTML την JavaScript και την CSS η δουλειά αυτού του vue component είναι να εμφανίζει μέσα από μια δομή επανάληψης όλα τα NoteBooks που υπάρχουν μέσα στην εφαρμογή καθώς και να μπορεί να στείλει τα αιτήματα επεξεργασίας και διαγραφής στον server για να είναι δυνατή η παραμετροποίηση και η διαγραφή δεδομένων. Η λογική εδώ είναι ότι κάθε φορά που φορτώσει το συγκεκριμένο component γίνεται αυτόματα ένα κάλεσμα προς τον

διακομιστή (server) ώστε να έρθουν όλα τα NoteBooks από την βάση δεδομένων και να εμφανιστούν στην εφαρμογή .Παρακάτω φαίνεται το παράδειγμα της διαγραφής ενός NoteBook με την βιβλιοθήκη axios που μας επιτρέπει να κάνουμε HTTP Requests πιο εύκολα και πιο γρήγορα.

Στην συνέχεια δημιουργούμε το Notes.vue το οποίο είναι το child component του NoteBooks Και ουσιαστικά εκεί εμφανίζονται όλα τα Notes για το κάθε NoteBook ξεχωριστά. Πάλι με μια δομή επανάληψης και κάποιους έλεγχους εμφανίζεται η λίστα με τα Notes τα οποία σε κάθε NoteBook μπορεί να είναι διαφορετικά και να είτε να υπάρχουν πολλά είτε λίγα είτε και κανένα. Σε αυτό το component υπάρχει η δυνατότητα να κάνουμε όλες τις δυνατές ενέργειες δηλαδή υπάρχει η δυνατότητα προσθήκης , επεξεργασίας αλλά και διαγραφής κάποιου Note καθώς για όλες αυτές τις ενέργειες υπάρχει η αντίστοιχη συνάρτηση η οποία κάνει το αίτημα στον διακομιστή (server) και μετά από εκεί παίρνει την απάντηση και ανάλογα με την απάντηση εμφανίζεται και το τελικό αποτέλεσμα στον χρήστη. οπότε έχουμε τον πλήρη έλεγχο για το συγκεκριμένο note ενός συγκεκριμένου NoteBook. Παρακάτω φαίνονται τα παραδείγματα της προσθήκης ενός Note και τον τρόπο με τον οποίο ξεχωρίζουμε και φέρνουμε από τον διακομιστή (server) ένα συγκεκριμένο note από κάποιο συγκεκριμένο NoteBook με το id του.

Το επόμενο component είναι το NoteContent.vue το οποίο είναι το περιεχόμενο που εμφανίζεται μέσα στο κάθε note μαζί με τις λειτουργίες επεξεργασία και διαγραφή. Στην δομή επανάληψης που γίνεται στο Note.vue component για κάθε επανάληψη αυτής της δομής παράγεται και αν NoteContent component καθώς μπορεί να υπάρχουν ένα , κανένα η και πολλά τέτοια components τα οποία είναι ξεχωριστά και μοναδικά και ανήκουν σε κάποιο συγκεκριμένο Note component.

Συμπεράσματα

Οι διαδικτυακές εφαρμογές στις μέρες μας γίνονται όλο και πιο δημοφιλείς και πλέον κάθε εταιρία και οργανισμός έχει τουλάχιστον από μια ιστοσελίδα είτε για την προβολή και διαφήμιση της επιχείρησης είτε για την διαχείριση δεδομένων μέσω web applications. Το web development με κάποιο από τα πολλά framework είναι η τελευταία λέξη της τεχνολογίας και βρίσκεται στο στάδιο συνεχούς αναβάθμισης και βελτίωσης. Σχεδόν καθημερινά πολλοί προγραμματιστές αναζητούν και ρωτούν ποιο είναι το καλύτερο framework για backend και ποιο για frontend. Η πιο απλή απάντηση είναι ότι το καλύτερο είναι αυτό που σε βοηθάει να φτιάξεις αυτό που θες γρήγορα εύκολα και να μπορεί να είναι επεκτάσιμο. Οπότε είναι καθαρά θέμα προσωπικής επιλογής. Έτσι και εγώ η επιλογή μου σε αυτά τα 2 framework (Laravel & Vue) έγινε επειδή θεωρώ προσωπικά ότι είναι κοντά στις γλώσσες προγραμματισμού που εγώ έχω ασχοληθεί και ποιο κοντά στα δικά μου γούστα στον τρόπο λειτουργίας και γραφής της εφαρμογής. Μέσω αυτής της πτυχιακής εργασίας είχα την δυνατότητα να μελετήσω σε βάθος κάποια σημεία αυτών των τεχνολογιών που τα ήξερα τόσο καλά η δεν τα θυμόμουν με τον σωστό τρόπο. Μελέτησα σε βάθος αρκετές από τις δυνατότητες που προσφέρουν αυτά τα εργαλεία και με τις γνώσεις που ήδη είχα αλλά και με αυτές που απέκτησα μετά από την μελέτη ξεκίνησα να δημιουργώ την εφαρμογή. Φυσικά κάθε φορά που δεν θυμόμουν κάτι το documentation των framework ήταν εκεί να με βοηθήσει να βρω την απάντηση που έψαχνα.

Μελλοντική επέκταση

Για την ενίσχυση και εμβάθυνση αυτής της εργασίας θα μπορούσαν να προστεθούν 2 με 3 ακόμα δυνατότητες. Η πρώτη και ποιο σημαντική θα μπορούσε να είναι η Λειτουργιά “Forgot your password? ” με την οποία ο χρήστης θα μπορεί σε περίπτωση που ξεχάσει τον κωδικό πρόσβασης του να τον αλλάξει και να δώσει ένα καινούργιο. Κάτι τέτοιο είναι εφικτό με κάποιο email από το backend της εφαρμογής προς τον χρήστη με κάποιο σύνδεσμο που θα τον κατευθύνει σε μια συγκεκριμένη σελίδα για να κάνει την αλλαγή του password. Μια άλλη βελτίωση θα μπορούσε να είναι η επικοινωνία των χρηστών μεταξύ τους με κάποιο σύστημα συνομιλίας ώστε να μπορούν οι χρήστες να μοιράζονται τις εικόνες, τα βίντεο και τις σημειώσεις τους και ακόμα καλύτερα να μπορούν να μοιραστούν μια ολόκληρη κατηγορία από κάποιες σημειώσεις. Τέλος μια ακόμα αναβάθμιση θα μπορούσε να είναι μια ενδεχόμενη διχρωμία της εφαρμογής και η εναλλαγή από άσπρο σε μαύρο ώστε να κάνει την ανάγνωση τις βραδινές ώρες πιο ξεκούραστη.

Βιβλιογραφία

1) Πτυχιακή εργασία με θέμα: "Καταχώρηση βλαβών και συμβάντων σε επίπεδο δήμου μέσω διαδικτυακής εφαρμογής" ---ΤΕΙ ΠΕΙΡΑΙΑ Τμήμα Υπολογιστικών Συστημάτων.

2) Γιάτας Δ. , Γώγουλος Γ. , Κοτίνη Ι. , Κυριακάκης Γ. , Μωράκης Δ. , Τζελέπης Σ. , Φραγκονικολάκης Μ. , Συστήματα Διαχείρισης Βάσεων Δεδομένων και Εφαρμογές τους στο Διαδίκτυο Β΄ Τάξη ΕΠΑ.Λ. 2016 , ΥΠΟΥΡΓΕΙΟ ΠΟΛΙΤΙΣΜΟΥ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ ΙΝΣΤΙΤΟΥΤΟ ΕΚΠΑΙΔΕΥΤΙΚΗΣ ΠΟΛΙΤΙΚΗΣ, ΙΝΣΤΙΤΟΥΤΟ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ & ΕΚΔΟΣΕΩΝ «ΔΙΟΦΑΝΤΟΣ» .

3) Zell Liew, J*anuary 17, 2018* , “Understanding And Using REST APIs”,[Online]: https://www.smashingmagazine.com/2018/01/understanding-using-rest-api

4) Sagara Technology Idea Lab, J*anuary 18, 2020* , “What Languages are Used for BackEndDevelopment?”,[Online]: https://sagaratechnology.medium.com/what-languages-are-used-for-back-end-development-71a8a10c135c

5) Nicole Ferguson, J*anuary 11, 2021* , “What's The Difference Between Frontend And Backend Web Development?”,[Online]: https://careerfoundry.com/en/blog/web-development/whats-the-difference-between-frontend-and-backend/#:~:text=Some%20common%20backend%20languages%20are,a%20framework%20written%20in%20Ruby.

6) Kyriakidis, A., & Maniatis, K. (2016). *The Majesty of Vue. js*. Packt Publishing Ltd.

7) Macrae, C. (2018). *Vue. js: Up and Running: Building Accessible and Performant Web Apps*. " O'Reilly Media, Inc.".
8) Wohlgethan, E. (2018). *Supportingweb development decisions by comparing three major javascript frameworks: Angular, react and vue. Js* (Doctoral dissertation, Hochschule für Angewandte Wissenschaften Hamburg).

9) Hossain, M. S. (2019). Web application development with Laravel framework.

10) Bean, M. (2015). *Laravel 5 essentials*. Packt Publishing Ltd.

11) Stauffer, M. (2019). *Laravel: Up & running: A framework for building modern php apps*. O'Reilly Media.

12) Saks, E. (2019). JavaScript Frameworks: Angular vs React vs Vue.

13) Masse, M. (2011). *REST API design rulebook: designing consistent RESTful web service interfaces*. " O'Reilly Media, Inc.".

14) Teixeira, P. (2012). *Professional Node. js: Building Javascript based scalable software*. John Wiley & Sons.

15) Crockford, D. (2008). *JavaScript: The Good Parts: The Good Parts*. " O'Reilly Media, Inc.".

16) Goodman, D. (2002). *Dynamic HTML: The definitive reference: A comprehensive resource for HTML, CSS, DOM & JavaScript*. " O'Reilly Media, Inc.".

Παράρτημα Α. Κώδικας PHP Laravel (Backend)

Αρχείο database → migrations → user Table

```php
<?php


    use Illuminate\Database\Migrations\Migration;
    use Illuminate\Database\Schema\Blueprint;
    use Illuminate\Support\Facades\Schema;

    class CreateUsersTable extends Migration
    {
        /**
         * Run the migrations.
         *
         * @return void
         */
        public function up()
        {
            Schema::create('users', function (Blueprint $table) {
                $table->id();
                $table->string('name');
                $table->string('email')->unique();
                $table->timestamp('email_verified_at')->nullable();
                $table->string('password');
                $table->rememberToken();
                $table->timestamps();
            });
        }

        /**
         * Reverse the migrations.
         *
         * @return void
         */
        public function down()
        {
            Schema::dropIfExists('users');
        }
    }
```

Αρχείο database → migrations → NoteBook Table

```php
<?php

    use Illuminate\Database\Migrations\Migration;
    use Illuminate\Database\Schema\Blueprint;
    use Illuminate\Support\Facades\Schema;

    class CreateNoteBooksTable extends Migration
    {
        /**
         * Run the migrations.
         *
         * @return void
         */
        public function up()
        {
            Schema::create('note_books', function (Blueprint $table) {
                $table->id();
                $table->unsignedBigInteger('user_id');
                $table->string('title');
                $table->timestamps();

                $table->foreign('user_id')->references('id')->on('users')->onDelete('cascade');
            });
        }

        /**
         * Reverse the migrations.
         *
         * @return void
         */
        public function down()
        {
            Schema::dropIfExists('note_books');
        }
    }
```

Αρχείο database → migrations → Notes Table

```
<?php


    use Illuminate\Database\Migrations\Migration;
    use Illuminate\Database\Schema\Blueprint;
    use Illuminate\Support\Facades\Schema;

    class CreateNotesTable extends Migration
    {
        /**
         * Run the migrations.
         *
         * @return void
         */
        public function up()
        {
            Schema::create('notes', function (Blueprint $table) {
                $table->id();
                $table->unsignedBigInteger('note_book_id');
                $table->string('title');
                $table->text('content')->nullable();
                $table->timestamps();

                $table->foreign('note_book_id')->references('id')->on('note_books')-
>onDelete('cascade');
            });
        }

        /**
         * Reverse the migrations.
         *
         * @return void
         */
        public function down()
        {
            Schema::dropIfExists('notes');
        }
    }
```

Αρχείο app → User.php Model

```php
<?php

    namespace App;

    use Illuminate\Contracts\Auth\MustVerifyEmail;
    use Illuminate\Foundation\Auth\User as Authenticatable;
    use Illuminate\Notifications\Notifiable;

    class User extends Authenticatable
    {
        use Notifiable;

        /**
         * The attributes that are mass assignable.
         *
         * @var array
         */
        protected $fillable = [
            'name', 'email', 'password',
        ];

        /**
         * The attributes that should be hidden for arrays.
         *
         * @var array
         */
        protected $hidden = [
            'password', 'remember_token',
        ];

        /**
         * The attributes that should be cast to native types.
         *
         * @var array
         */
        protected $casts = [
            'email_verified_at' => 'datetime',
        ];

        public function getNotebookNameAttribute()
        {
            return ucfirst($this->name) . '\'s Notebooks';
```

```
    }

    public function notebooks()
    {
        return $this->hasMany(NoteBook::class);
    }
}
```

Αρχείο app → NoteBook.php Model

```
<?php

    namespace App;

    use Illuminate\Database\Eloquent\Model;

    class NoteBook extends Model
    {
        public static function booted()
        {
            static::deleting(function ($notebook) {
                $notebook->notes()->delete();
            });
        }

        protected $fillable = [
            'user_id', 'title'
        ];

        protected $hidden = [
            'user_id', 'created_at', 'updated_at'
        ];

        public function user()
        {
            return $this->belongsTo(User::class);
        }

        public function notes()
```

```
    {
        return $this->hasMany(Note::class);
    }

    public static function userNotebooks()
    {
        return static::where('user_id', auth()->id())->latest()->get();
    }
}
```

Αρχείο app → Note.php Model

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Note extends Model
{
    protected $fillable = [
        'note_book_id', 'title', 'content'
    ];

    protected $hidden = [
        'created_at', 'updated_at'
    ];

    protected $appends = [
        'date', 'time'
    ];

    public function getDateAttribute()
    {
        return $this->created_at->format('d F Y');
    }

    public function getTimeAttribute()
    {
```

```
            return $this->created_at->format('h:ia');
        }

        public function notebook()
        {
            return $this->belongsTo(NoteBook::class);
        }
    }
```

Αρχείο database → factories -> Userfactory.php

```
<?php

    use App\User;
    use Faker\Generator as Faker;
    use Illuminate\Support\Str;

    $factory->define(User::class, function (Faker $faker) {
        return [
            'name' => $faker->name,
            'email' => $faker->unique()->safeEmail,
            'email_verified_at' => now(),
            'password' =>
'$2y$10$92IXUNpkjO0rOQ5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi', // password
            'remember_token' => Str::random(10),
        ];
    });
```

Αρχείο database → factories -> NoteBookFactory.php

```
    <?php

    use App\NoteBook;
    use Faker\Generator as Faker;

    $factory->define(NoteBook::class, function (Faker $faker) {
        return [
            'title' => $faker->name
```

```
        ];
    });
```

Αρχείο database → factories -> NoteFactory.php

```
<?php

use App\Note;
use Faker\Generator as Faker;

$factory->define(Note::class, function (Faker $faker) {
    return [
        'title' => $faker->name,
        'content' => '<p>' . $faker->paragraph(20) . '</p>' . '<p>' . $faker->paragraph(20)
. '</p>'
    ];
});
```

Αρχείο database → seeds -> UserSeeder.php

```
<?php

use App\User;
use Illuminate\Database\Seeder;

class UserSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        User::create([
            'name' => 'User',
            'email' => 'user@test.com',
            'password' => bcrypt('password')
        ]);
```

```
    }
}
```

Αρχείο database → seeds -> NoteBookSeeder.php

```
<?php

use App\NoteBook;
use App\User;
use Illuminate\Database\Seeder;

class NoteBookSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $user = User::first();

        $notebooks = factory(NoteBook::class, 10)->make();

        $user->notebooks()->saveMany($notebooks);
    }
}
```

Αρχείο database → seeds -> NoteSeeder.php

```
<?php

use App\NoteBook;
use App\Note;
use Illuminate\Database\Seeder;

class NoteSeeder extends Seeder
```

```
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $notebooks = NoteBook::take(10)->get();

        foreach ($notebooks as $notebook) {
            $notes = factory(Note::class, 10)->make();

            $notebook->notes()->saveMany($notes);
        }
    }
}
```

Αρχείο database → seeds -> DatabaseSeeder.php

```
<?php

use App\Note;
use App\NoteBook;
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     *
     * @return void
     */
    public function run()
    {
        $this->call([
            UserSeeder::class,
            NoteBookSeeder::class,
            NoteSeeder::class
        ]);
    }
```

```
}
```

Αρχείο database → config -> auth.php

```
<?php

return [

    'defaults' => [
        'guard' => 'web',
        'passwords' => 'users',
    ],

    'guards' => [
        'web' => [
            'driver' => 'session',
            'provider' => 'users',
        ],

        'api' => [
            'driver' => 'token',
            'provider' => 'users',
            'hash' => false,
        ],
    ],

    'providers' => [
        'users' => [
            'driver' => 'eloquent',
            'model' => App\User::class,
        ],

    ],

    'passwords' => [
        'users' => [
            'provider' => 'users',
            'table' => 'password_resets',
            'expire' => 60,
            'throttle' => 60,
        ],
    ],
```

```
    'password_timeout' => 10800,

];
```

Αρχείο database → config -> database.php

```
<?php

use Illuminate\Support\Str;

return [

    'default' => env('DB_CONNECTION', 'mysql'),
    'connections' => [

        'sqlite' => [
            'driver' => 'sqlite',
            'url' => env('DATABASE_URL'),
            'database' => env('DB_DATABASE', database_path('database.sqlite')),
            'prefix' => '',
            'foreign_key_constraints' => env('DB_FOREIGN_KEYS', true),
        ],

        'mysql' => [
            'driver' => 'mysql',
            'url' => env('DATABASE_URL'),
            'host' => env('DB_HOST', '127.0.0.1'),
            'port' => env('DB_PORT', '3306'),
            'database' => env('DB_DATABASE', 'forge'),
            'username' => env('DB_USERNAME', 'forge'),
            'password' => env('DB_PASSWORD', ''),
            'unix_socket' => env('DB_SOCKET', ''),
            'charset' => 'utf8mb4',
            'collation' => 'utf8mb4_unicode_ci',
            'prefix' => '',
            'prefix_indexes' => true,
            'strict' => true,
            'engine' => null,
            'options' => extension_loaded('pdo_mysql') ? array_filter([
                PDO::MYSQL_ATTR_SSL_CA => env('MYSQL_ATTR_SSL_CA'),
```

```
            ]) : [],
        ],

        'pgsql' => [
            'driver' => 'pgsql',
            'url' => env('DATABASE_URL'),
            'host' => env('DB_HOST', '127.0.0.1'),
            'port' => env('DB_PORT', '5432'),
            'database' => env('DB_DATABASE', 'forge'),
            'username' => env('DB_USERNAME', 'forge'),
            'password' => env('DB_PASSWORD', ''),
            'charset' => 'utf8',
            'prefix' => '',
            'prefix_indexes' => true,
            'schema' => 'public',
            'sslmode' => 'prefer',
        ],

        'sqlsrv' => [
            'driver' => 'sqlsrv',
            'url' => env('DATABASE_URL'),
            'host' => env('DB_HOST', 'localhost'),
            'port' => env('DB_PORT', '1433'),
            'database' => env('DB_DATABASE', 'forge'),
            'username' => env('DB_USERNAME', 'forge'),
            'password' => env('DB_PASSWORD', ''),
            'charset' => 'utf8',
            'prefix' => '',
            'prefix_indexes' => true,
        ],

    ],

    'migrations' => 'migrations',
    'redis' => [

        'client' => env('REDIS_CLIENT', 'phpredis'),

        'options' => [
            'cluster' => env('REDIS_CLUSTER', 'redis'),
            'prefix' => env('REDIS_PREFIX', Str::slug(env('APP_NAME', 'laravel'),
'_').'_database_'),
        ],
```

```
        'default' => [
            'url' => env('REDIS_URL'),
            'host' => env('REDIS_HOST', '127.0.0.1'),
            'password' => env('REDIS_PASSWORD', null),
            'port' => env('REDIS_PORT', '6379'),
            'database' => env('REDIS_DB', '0'),
        ],

        'cache' => [
            'url' => env('REDIS_URL'),
            'host' => env('REDIS_HOST', '127.0.0.1'),
            'password' => env('REDIS_PASSWORD', null),
            'port' => env('REDIS_PORT', '6379'),
            'database' => env('REDIS_CACHE_DB', '1'),
        ],

    ],

];
```

Αρχείο database → config -> cors.php

```
<?php

return [
    'paths' => ['api/*'],

    'allowed_methods' => ['*'],

    'allowed_origins' => ['*'],

    'allowed_origins_patterns' => [],

    'allowed_headers' => ['*'],

    'exposed_headers' => [],

    'max_age' => 0,

    'supports_credentials' => false,

];
```

Αρχείο database → config -> session.php

```php
<?php

use Illuminate\Support\Str;

return [

    'driver' => env('SESSION_DRIVER', 'file'),

    'lifetime' => env('SESSION_LIFETIME', 120),

    'expire_on_close' => false,

    'encrypt' => false,

    'files' => storage_path('framework/sessions'),

    'connection' => env('SESSION_CONNECTION', null),

    'table' => 'sessions',

    'store' => env('SESSION_STORE', null),

    'lottery' => [2, 100],

    'cookie' => env(
        'SESSION_COOKIE',
        Str::slug(env('APP_NAME', 'laravel'), '_').'_session'
    ),

    'path' => '/',

    'domain' => env('SESSION_DOMAIN', null),

    'secure' => env('SESSION_SECURE_COOKIE'),

    'http_only' => true,

    'same_site' => 'lax',

];
```

Αρχείο database → Exceptions -> Handlers.php

```php
<?php

    namespace App\Exceptions;

    use Illuminate\Foundation\Exceptions\Handler as ExceptionHandler;
    use Throwable;

    class Handler extends ExceptionHandler
    {
        /**
         * A list of the exception types that are not reported.
         *
         * @var array
         */
        protected $dontReport = [
            //
        ];

        /**
         * A list of the inputs that are never flashed for validation exceptions.
         *
         * @var array
         */
        protected $dontFlash = [
            'password',
            'password_confirmation',
        ];

        /**
         * Report or log an exception.
         *
         * @param  \Throwable  $exception
         * @return void
         *
         * @throws \Exception
         */
        public function report(Throwable $exception)
        {
            parent::report($exception);
        }
```

```
    /**
     * Render an exception into an HTTP response.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Throwable  $exception
     * @return \Symfony\Component\HttpFoundation\Response
     *
     * @throws \Throwable
     */
    public function render($request, Throwable $exception)
    {
        return parent::render($request, $exception);
    }
}
```

Αρχείο database → Middleware -> RedirectIfAuthenticated.php

```
<?php

namespace App\Http\Middleware;

use App\Providers\RouteServiceProvider;
use Closure;
use Illuminate\Support\Facades\Auth;

class RedirectIfAuthenticated
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @param  string|null  $guard
     * @return mixed
     */
    public function handle($request, Closure $next, $guard = null)
    {
        if (Auth::guard($guard)->check()) {
            return redirect(RouteServiceProvider::HOME);
```

```
        }

        return $next($request);
    }
}
```

Αρχείο database → Middleware → Authenticate.php

```
<?php

namespace App\Http\Middleware;

use Illuminate\Auth\Middleware\Authenticate as Middleware;

class Authenticate extends Middleware
{
    /**
     * Get the path the user should be redirected to when they are not authenticated.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return string|null
     */
    protected function redirectTo($request)
    {
        if (! $request->expectsJson()) {
            return route('login');
        }
    }
}
```

Αρχείο app → controllers → Auth → RegisterController.php

```
<?php

namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;
use App\Providers\RouteServiceProvider;
use App\User;
use Illuminate\Foundation\Auth\RegistersUsers;
```

```
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Validator;

class RegisterController extends Controller
{
    /*
    |--------------------------------------------------------------------------
    | Register Controller
    |--------------------------------------------------------------------------
    |
    | This controller handles the registration of new users as well as their
    | validation and creation. By default this controller uses a trait to
    | provide this functionality without requiring any additional code.
    |
    */

    use RegistersUsers;

    /**
     * Where to redirect users after registration.
     *
     * @var string
     */
    protected $redirectTo = RouteServiceProvider::HOME;

    /**
     * Create a new controller instance.
     *
     * @return void
     */
    public function __construct()
    {
        $this->middleware('guest');
    }

    /**
     * Get a validator for an incoming registration request.
     *
     * @param  array  $data
     * @return \Illuminate\Contracts\Validation\Validator
     */
    protected function validator(array $data)
    {
        return Validator::make($data, [
```

```
            'name' => ['required', 'string', 'max:255'],
            'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
            'password' => ['required', 'string', 'min:8', 'confirmed'],
        ]);
    }

    /**
     * Create a new user instance after a valid registration.
     *
     * @param  array  $data
     * @return \App\User
     */
    protected function create(array $data)
    {
        return User::create([
            'name' => $data['name'],
            'email' => $data['email'],
            'password' => Hash::make($data['password']),
        ]);
    }
}
```

Αρχείο app → controllers → NoteBookController.php

```
<?php

namespace App\Http\Controllers;

use App\NoteBook;
use Illuminate\Http\Request;

class NoteBookController extends Controller
{
    /**
     * Create a new controller instance.
     *
     * @return void
     */
    public function __construct()
    {
        $this->middleware('auth');
    }
```

```php
/**
 * Display a listing of the resource.
 *
 * @return \Illuminate\Http\Response
 */
public function index(Request $request)
{
    if ($request->ajax() || $request->isJson()) {
        return response()->json(NoteBook::userNotebooks(), 200);
    }

    return view('home');
}

/**
 * Show the form for creating a new resource.
 *
 * @return \Illuminate\Http\Response
 */
public function create()
{
    return view('notebook.create');
}

/**
 * Store a newly created resource in storage.
 *
 * @param  \Illuminate\Http\Request  $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{
    $request->validate(['title' => 'required|unique:note_books,title']);

    NoteBook::create([
        'title' => $request->title,
        'user_id' => auth()->id()
    ]);

    return redirect()->route('notebook.index');
}

/**
```

```
     * Display the specified resource.
     *
     * @param  \App\NoteBook  $notebook
     * @return \Illuminate\Http\Response
     */
    public function show(NoteBook $notebook)
    {
        $notes = $notebook->notes()->get();

        return response()->json($notes, 200);
    }

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \App\NoteBook  $noteBook
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, NoteBook $notebook)
    {
        $v = validator($request->all(), ['title' => 'required']);

        if($v->fails()) return response()->json($v->errors()->all(), 422);

        $notebook->update([
            'user_id' => auth()->id(),
            'title' => $request->title
        ]);

        return response()->json(true, 200);
    }

    /**
     * Notebook Image Upload
     *
     * @param Request $request
     * @return string $url
     */
    public function upload(Request $request)
    {
        $v = validator($request->all(), ['upload' => 'required']);

        if($v->fails()) {
```

```
            return response()->json([
                'error' => ['message' => $v->errors()->first()]
            ]);
        }

        $url = $request->upload->store('uploads', 'public');

        $url = asset("storage/$url");

        return response()->json(compact("url"), 200);
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  \App\NoteBook  $noteBook
     * @return \Illuminate\Http\Response
     */
    public function destroy(NoteBook $notebook)
    {
        $notebook->delete();

        return response()->json(true, 200);
    }
}
```

Αρχείο app → controllers → NoteController.php

```
<?php

namespace App\Http\Controllers;

use App\Note;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Validator;

class NoteController extends Controller
{
    /**
     * Create a new controller instance.
     *
```

```
     * @return void
     */
    public function __construct()
    {
        $this->middleware('auth');
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        $v = Validator::make($request->all(), ['title' => 'required']);

        if($v->fails()) return response()->json($v->errors()->all(), 422);

        $note = Note::create([
            'title' => $request->title,
            'note_book_id' => $request->note_book_id
        ]);

        return response()->json($note, 200);
    }

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \App\Note  $note
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, Note $note)
    {
        $v = Validator::make($request->all(), [
            'title' => 'required_without:content',
            'content' => 'required_without:title',
        ]);

        if($v->fails()) return response()->json($v->errors()->all(), 422);

        $note->update($v->valid());
```

```
        return response()->json(true, 200);
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  \App\Note  $note
     * @return \Illuminate\Http\Response
     */
    public function destroy(Note $note)
    {
        $note->delete();

        return response()->json(true);
    }
}
```

Αρχείο app → routes → api.php

```
<?php

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

Route::middleware('auth:api')->get('/user', function (Request $request) {
    return $request->user();
});
```

Αρχείο app → routes → web.php

```
<?php

use Illuminate\Support\Facades\Route;
use Illuminate\Support\Facades\Auth;

Auth::routes();

Route::view('/', 'welcome');

Route::resource('notebook', 'NotebookController');

Route::resource('note', 'NoteController')->only(['store', 'update', 'destroy']);
```

```
Route::post('upload', 'NoteBookController@upload');
```

Παράρτημα Β. Κώδικας JavaScript, HTML, SCSS, Vue (Frontend)

Αρχείο app → resources → js → app.js

```
require('./bootstrap');

window.Vue = require('vue');

Vue.component('notebooks', require('./components/Notebooks.vue').default);

const app = new Vue({
    el: '#app',
});
```

Αρχείο app → resources → js → bootstrap.js

```
window._ = require('lodash');

try {
    window.Popper = require('popper.js').default;
    window.$ = window.jQuery = require('jquery');

    require('bootstrap');
} catch (e) {}

window.axios = require('axios');

window.axios.defaults.headers.common['X-Requested-With'] = 'XMLHttpRequest';
window.axios.defaults.headers.common['X-CSRF-TOKEN'] = window.csrf;
```

Αρχείο webpack.mix.js

```
const mix = require('laravel-mix');
```

```
mix.js('resources/js/app.js', 'public/js')
   .sass('resources/sass/app.scss', 'public/css');
```

Αρχείο app → resources → js → components → NoteBooks.vue

```
<template>
   <div class="container-fluid">
      <div class="row" v-if="notebooks.length">
         <div class="col-md-3">
            <div class="list-group sticky-top">
               <button
                  class="list-group-item list-group-item-action notebook"
                  :class="activeNotebook.id == notebook.id ? 'active' : ""
                  v-for="notebook in notebooks"
                  :key="notebook.id"
                  @click="change(notebook)"
               >
                  <span class="edit" v-if="isActiveEdit(notebook)">
                     <input
                        type="text"
                        :id="activeNoteBookEditId"
                        v-model="activeNoteBookEditTitle"
                        v-on:keyup.enter="updateActiveEdit"
                     />

                     <div class="actions">
                        <i class="ti-close mr-2" @click="closeActiveEdit"></i>
                        <i class="ti-check" @click="updateActiveEdit"></i>
                     </div>
                  </span>

                  <span class="text" v-else>
                     <i class="ti-notepad mr-2"></i>{{ notebook.title }}

                     <ul class="list-inline m-0 float-right hidden">
                        <li class="list-inline-item">
                           <i class="ti-pencil" @click="edit(notebook)"></i>
                        </li>
                        <li class="list-inline-item text-danger">
                           <i class="ti-trash" @click="destroy(notebook)"></i>
```

```
                        </li>
                    </ul>
                </span>
            </button>
        </div>
    </div>

    <notes v-if="activeNotebook" :notebook="activeNotebook"></notes>
</div>

<div class="row" v-else>
    <div class="col-12 d-flex justify-content-center flex-column mt-5" >
        <h1 class="text-center">
            <i class="ti-pencil-alt d-block"></i>
            Welcome to Laravel Notebook! <br>
        </h1>
        <p class="text-center text-muted">Lets get started with a new
Notebook.</p>
    </div>
</div>
</div>
</template>

<script>
import Note from "./Notes";

export default {
    data() {
        return {
            notebooks: [],
            activeNotebook: null,
            activeNoteBookEdit: null,
            activeNoteBookEditTitle: null
        };
    },
    components: {
        notes: Note
    },
    methods: {
        get() {
            axios
                .get("/notebook")
                .then(response => {
                    this.notebooks = response.data;
```

```
                this.activeNotebook = this.notebooks[0];
            })
            .catch(error => {
                console.log(error);
            });
        },
        edit(notebook) {
            this.activeNoteBookEdit = notebook;
            this.activeNoteBookEditTitle = notebook.title;
            this.$nextTick(() =>
                document.getElementById(this.activeNoteBookEditId).focus()
            );
        },
        destroy(notebook) {
            axios
                .delete(`/notebook/${notebook.id}`)
                .then(response => {
                    this.notebooks = this.notebooks.filter(
                        el => el.id !== notebook.id
                    );
                    this.activeNotebook = this.notebooks[0];
                })
                .catch(error => {
                    console.log(error);
                });
        },
        change(notebook) {
            this.activeNotebook = notebook;

            if(this.activeNoteBookEdit && this.activeNotebook.id !==
this.activeNoteBookEdit.id) {
                this.closeActiveEdit();
            }
        },
        isActiveEdit(notebook) {
            return (
                this.activeNoteBookEdit &&
                notebook.id == this.activeNoteBookEdit.id
            );
        },
        updateActiveEdit() {
            if (!this.activeNoteBookEditTitle) return;

            if (this.activeNoteBookEdit.title == this.activeNoteBookEditTitle)
```

```
                return this.closeActiveEdit();

            let index = this.notebooks.findIndex(
                notebook => notebook.id == this.activeNoteBookEdit.id
            );

            axios
                .patch(`/notebook/${this.activeNoteBookEdit.id}`, {
                    title: this.activeNoteBookEditTitle
                })
                .then(response => {
                    this.notebooks[index].title = this.activeNoteBookEditTitle;
                })
                .catch(error => {
                    console.log(error);
                })
                .finally(() => {
                    this.closeActiveEdit();
                });
        },
        closeActiveEdit() {
            this.activeNoteBookEdit = null;
            this.activeNoteBookEditTitle = null;
        }
    },
    computed: {
        activeNoteBookEditId() {
            return `notebook-input-${this.activeNoteBookEdit.id}`;
        }
    },
    mounted() {
        this.get();
    }
};
</script>
```

Αρχείο app → resources → js → components → Notes.vue

```
<template>
    <div class="col-md-9">
        <div class="row">
```

```
            <note-content v-if="activeNote" :note="activeNote"
@deleted="deleted"></note-content>

            <div class="col-md-4">
              <div class="card sticky-top">
                <div class="card-header">
                  <a href="#!" @click="add">
                    <i class="ti-plus"></i> Add Note
                  </a>
                </div>

                <div class="card-body">
                  <ul class="list-group list-group-flush" v-if="notes.length">
                    <button
                      class="list-group-item text-left"
                      v-for="note in notes"
                      :key="note.id"
                      :class="activeNote.id == note.id ? 'active' : ""
                      @click="change(note)"
                    >{{ note.title }}</button>
                  </ul>
                </div>
              </div>
            </div>
          </div>
        </template>

        <script>
        import NoteContent from "./NoteContent";

        export default {
          props: ["notebook"],
          data() {
            return {
              notes: [],
              activeNote: null,
              defaultNoteTitle: "Untitled Note"
            };
          },
          components: {
            "note-content": NoteContent
          },
          watch: {
```

```
    notebook: {
      handler: function(notebook) {
        this.get();
      }
    }
  },
  methods: {
    get() {
      axios
        .get(`/notebook/${this.notebook.id}`)
        .then(response => {
          this.notes = response.data;
          this.activeNote = this.notes[0];
        })
        .catch(error => {
          console.log(error);
        });
    },
    add() {
      axios
        .post("/note", {
          title: this.defaultNoteTitle,
          note_book_id: this.notebook.id
        })
        .then(response => {
          let note = response.data;

          this.activeNote = note;
          this.notes.push(note);
        })
        .catch(error => {
          console.log(error);
        });
    },
    change(note) {
      this.activeNote = note;
    },
    deleted() {
      this.notes = this.notes.filter(
        note => note.id !== this.activeNote.id
      );
      this.activeNote = this.notes[0];
    }
  },
```

```
    mounted() {
      if(this.notebook) this.get();
    }
};
</script>
```

Αρχείο app → resources → js → components → Notes.vue

```
<template>
  <div class="col-md-8">
    <div class="card">
      <div class="card-body">
        <h1 class="card-title border-bottom title">
          <span class="edit" v-if="titleEdit">
            <input
              type="text"
              id="note-content-title"
              v-model="newTitle"
              v-on:keyup.enter="updateNoteTitle"
            />

            <div class="actions">
              <i class="ti-close text-danger mr-2" @click="closeTitleEdit"></i>
              <i class="ti-check text-success" @click="updateNoteTitle"></i>
            </div>
          </span>

          <span class="text" v-else>
            {{ note.title }}
            <i
              class="ti-pencil text-primary float-right"
              @click="editTitle"
            ></i>
          </span>
        </h1>

        <ul class="list-inline">
          <li class="list-inline-item text-muted">{{ note.date }}</li>
          <li class="list-inline-item text-muted">{{ note.time }}</li>

          <li class="list-inline-item float-right" v-if="contentEdit">
```

```
                    <a href="#!" @click="updateNoteContent" class="mr-2">
                      <i class="ti-save"></i> Save Edited Content
                    </a>

                    <a href="#!" @click="destroyContentEditor" class="text-danger">
                      <i class="ti-eraser"></i> Cancel Editing
                    </a>
                  </li>

                  <li class="list-inline-item float-right" v-else>
                    <a href="#!" class="text-primary" @click="createContentEditor">
                      <i class="ti-eraser"></i> Edit note
                    </a>
                  </li>
                </ul>

                <div v-html="note.content" id="editor" ref="content" class="ck-
content"></div>

                <div class="mt-4">
                  <a href="#!" @click="deleteNote" class="text-danger">
                    <i class="ti-trash"></i> I confirm to delete this note!
                  </a>
                </div>
              </div>
            </div>
          </div>
        </template>

        <script>

        require("../ckeditor5-build-balloon-block/build/ckeditor");

        export default {
          props: ["note"],
          data() {
            return {
              titleEdit: false,
              contentEdit: false,
              contentEditor: null,
              newTitle: null,
              editorOptions: {
                placeholder: 'Start Writing Content...',
                image: {
```

```
                toolbar: ['imageTextAlternative', '|', 'imageStyle:alignLeft',
'imageStyle:full', 'imageStyle:alignRight'],
                styles: ['full', 'alignLeft', 'alignRight']
            },
            mediaEmbed: {
                previewsInDataData: true
            },
            simpleUpload: {
                // The URL that the images are uploaded to.
                uploadUrl: '/upload',

                // Headers sent along with the XMLHttpRequest to the upload server.
                headers: {
                    'X-CSRF-TOKEN': window.csrf
                }
            }
        }
    };
},
watch: {
    note: {
        handler: function(note) {
            this.newTitle = this.note ? this.note.title : null;
            this.closeTitleEdit();
            this.destroyContentEditor();
        }
    }
},
methods: {
    editTitle() {
        this.titleEdit = true;
        this.$nextTick(() =>
            document.getElementById("note-content-title").focus()
        );
    },
    createContentEditor() {
        if (!this.contentEditor) {
            this.contentEdit = true;

            BalloonEditor.create(document.querySelector("#editor"),
this.editorOptions)
                .then(editor => {
                    this.contentEditor = editor;
                })
```

```
            .catch(error => {
                console.error(error);
            })
            .finally(() => {
                this.contentEditor.editing.view.focus();
            });
        }
    },
    destroyContentEditor() {
        if (this.contentEditor) {
            this.contentEditor
                .destroy()
                .catch(error => {
                    console.log(error);
                })
                .finally(() => {
                    this.contentEdit = false;
                    this.contentEditor = null;
                    this.$refs["content"].innerHTML = this.note ? this.note.content : null;
                });
        }
    },
    closeTitleEdit() {
        this.titleEdit = false;
        this.newTitle = this.note ? this.note.title : null;
    },
    updateNoteTitle() {
        if (this.newTitle == this.note.title) return this.closeTitleEdit();

        axios
            .patch(`/note/${this.note.id}`, { title: this.newTitle })
            .then(response => {
                this.note.title = this.newTitle;
            })
            .catch(error => {
                console.log(error);
            })
            .finally(() => {
                this.closeTitleEdit();
            });
    },
    updateNoteContent(e) {
        let contentData = this.contentEditor.getData();
```

```
            if (contentData == this.note.content)
                return this.destroyContentEditor();

            axios
                .patch(`/note/${this.note.id}`, { content: contentData })
                .then(response => {
                    if(contentData) this.note.content = contentData;
                })
                .catch(error => {
                    console.log(error);
                })
                .finally(() => {
                    this.destroyContentEditor();
                });
        },
        deleteNote() {
            axios
                .delete(`/note/${this.note.id}`)
                .then(response => {
                    this.$emit("deleted");
                })
                .catch(error => {
                    console.log(error);
                });
        }
    }
};
</script>
```

Αρχείο app → resources → views → layouts → app.blade.php

```
<!doctype html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- CSRF Token -->
    <meta name="csrf-token" content="{{ csrf_token() }}">

    <title>{{ config('app.name', 'Laravel') }}</title>
```

```
        <!-- Scripts -->
        <script src="{{ asset('js/app.js') }}" defer></script>

        <!-- Fonts -->
        <link rel="dns-prefetch" href="//fonts.gstatic.com">
        <link href="https://fonts.googleapis.com/css?family=Nunito" rel="stylesheet">

        <!-- Styles -->
        <link href="{{ asset('css/app.css') }}" rel="stylesheet">

        <!-- Icons -->
        <link href="{{ asset('themify/themify-icons.css') }}" rel="stylesheet">

        <!-- Scripts -->
        <script> window.csrf = "{{ csrf_token() }}"; </script>
    </head>
    <body>
        <div id="app">
            @include('includes.navbar')

            <main class="py-4">
                @yield('content')
            </main>
        </div>
    </body>
    </html>
```

Αρχείο app → resources → views → includes → navbar.blade.php

```
    <nav class="navbar navbar-expand-md navbar-light bg-white shadow-sm">
        <div class="container-fluid">
            <a class="navbar-brand" href="{{ url('/') }}">
                {{ config('app.name', 'Laravel') }}
            </a>
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-
expanded="false" aria-label="{{ __('Toggle navigation') }}">
                <span class="navbar-toggler-icon"></span>
```

```
            </button>

            <div class="collapse navbar-collapse" id="navbarSupportedContent">
                <!-- Left Side Of Navbar -->
                <ul class="navbar-nav mr-auto">

                </ul>

                <!-- Right Side Of Navbar -->
                <ul class="navbar-nav ml-auto">
                    <!-- Authentication Links -->
                    @guest
                        <li class="nav-item">
                            <a class="nav-link" href="{{ route('login') }}">{{ __('Login') }}</a>
                        </li>
                        @if (Route::has('register'))
                            <li class="nav-item">
                                <a class="nav-link"
href="{{ route('register') }}">{{ __('Register') }}</a>
                            </li>
                        @endif
                    @else
                        <li class="nav-item dropdown">
                            <a id="navbarDropdown" class="nav-link dropdown-toggle" href="#"
role="button" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false" v-
pre>
                                {{ Auth::user()->name }} <span class="caret"></span>
                            </a>

                            <div class="dropdown-menu dropdown-menu-right" aria-
labelledby="navbarDropdown">
                                <a class="dropdown-item" href="{{ route('logout') }}"
                                    onclick="event.preventDefault();
                                                document.getElementById('logout-form').submit();">
                                    {{ __('Logout') }}
                                </a>

                                <form id="logout-form" action="{{ route('logout') }}"
method="POST" style="display: none;">
                                    @csrf
                                </form>
                            </div>
                        </li>
                    @endguest
```

```
            </ul>
          </div>
        </div>
    </nav>
```

Αρχείο app → resources → views → welcome.blade.php

```
    <!DOCTYPE html>
    <html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
      <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">

        <title>Laravel</title>

        <!-- Fonts -->
        <link href="https://fonts.googleapis.com/css?family=Nunito:200,600"
rel="stylesheet">

        <!-- Styles -->
        <style>
          html, body {
            background-color: #fff;
            color: #636b6f;
            font-family: 'Nunito', sans-serif;
            font-weight: 200;
            height: 100vh;
            margin: 0;
          }

          .full-height {
            height: 100vh;
          }

          .flex-center {
            align-items: center;
            display: flex;
            justify-content: center;
          }
```

```
            .position-ref {
                position: relative;
            }

            .top-right {
                position: absolute;
                right: 10px;
                top: 18px;
            }

            .content {
                text-align: center;
            }

            .title {
                font-size: 84px;
            }

            .links > a {
                color: #636b6f;
                padding: 0 25px;
                font-size: 13px;
                font-weight: 600;
                letter-spacing: .1rem;
                text-decoration: none;
                text-transform: uppercase;
            }

            .m-b-md {
                margin-bottom: 30px;
            }
        </style>
    </head>
    <body>
        <div class="flex-center position-ref full-height">
            @if (Route::has('login'))
                <div class="top-right links">
                    @auth
                        <a href="{{ route('notebook.index') }}">Home</a>
                    @else
                        <a href="{{ route('login') }}">Login</a>

                        @if (Route::has('register'))
                            <a href="{{ route('register') }}">Register</a>
```

```
                @endif
            @endauth
        </div>
    @endif

    <div class="content">
        <div class="title m-b-md">
            {{ config('app.name', 'Notebook') }}
        </div>

        <div class="links">
            <a href="{{ route('notebook.index') }}">Continue</a>
        </div>
    </div>
</div>
</body>
</html>
```

Αρχείο app → resources → views → home.blade.php

```
@extends('layouts.app')

@section('content')
    <div class="container-fluid">
        <div class="row justify-content-center">
            <div class="col-12 mb-4 d-flex align-items-center justify-content-between">
                <h4 class="m-0 text-muted font-weight-bold">
                    {{ auth()->user()->notebook_name }}
                </h4>

                <a href="{{ route('notebook.create') }}" class="btn btn-outline-primary">
                    <i class="ti-pencil-alt"></i> Create New Notebook
                </a>
            </div>
        </div>
    </div>

    <notebooks></notebooks>
@endsection
```

Αρχείο app → resources → views → auth → register.blade.php

```
@extends('layouts.app')

    @section('content')
    <div class="container">
       <div class="row justify-content-center">
          <div class="col-md-8">
             <div class="card">
                <div class="card-header">{{ __('Register') }}</div>

                <div class="card-body">
                   <form method="POST" action="{{ route('register') }}">
                      @csrf

                      <div class="form-group row">
                         <label for="name" class="col-md-4 col-form-label text-md-
right">{{ __('Name') }}</label>

                         <div class="col-md-6">
                            <input id="name" type="text" class="form-control
@error('name') is-invalid @enderror" name="name" value="{{ old('name') }}" required
autocomplete="name" autofocus>

                            @error('name')
                               <span class="invalid-feedback" role="alert">
                                  <strong>{{ $message }}</strong>
                               </span>
                            @enderror
                         </div>
                      </div>

                      <div class="form-group row">
                         <label for="email" class="col-md-4 col-form-label text-md-
right">{{ __('E-Mail Address') }}</label>

                         <div class="col-md-6">
                            <input id="email" type="email" class="form-control
@error('email') is-invalid @enderror" name="email" value="{{ old('email') }}" required
autocomplete="email">

                            @error('email')
                               <span class="invalid-feedback" role="alert">
                                  <strong>{{ $message }}</strong>
                               </span>
```

```
                    @enderror
                </div>
            </div>

            <div class="form-group row">
                <label for="password" class="col-md-4 col-form-label text-md-
right">{{ __('Password') }}</label>

                <div class="col-md-6">
                    <input id="password" type="password" class="form-control
@error('password') is-invalid @enderror" name="password" required
autocomplete="new-password">

                    @error('password')
                        <span class="invalid-feedback" role="alert">
                            <strong>{{ $message }}</strong>
                        </span>
                    @enderror
                </div>
            </div>

            <div class="form-group row">
                <label for="password-confirm" class="col-md-4 col-form-label text-
md-right">{{ __('Confirm Password') }}</label>

                <div class="col-md-6">
                    <input id="password-confirm" type="password" class="form-
control" name="password_confirmation" required autocomplete="new-password">
                </div>
            </div>

            <div class="form-group row mb-0">
                <div class="col-md-6 offset-md-4">
                    <button type="submit" class="btn btn-primary">
                        {{ __('Register') }}
                    </button>
                </div>
            </div>
        </form>
    </div>
</div>
</div>
</div>
```

```
    @endsection
```

Αρχείο app → resources → views → auth → login.blade.php

```
@extends('layouts.app')

    @section('content')
    <div class="container">
       <div class="row justify-content-center">
          <div class="col-md-8">
             <div class="card">
                <div class="card-header">{{ __('Login') }}</div>

                <div class="card-body">
                   <form method="POST" action="{{ route('login') }}">
                      @csrf

                      <div class="form-group row">
                         <label for="email" class="col-md-4 col-form-label text-md-
right">{{ __('E-Mail Address') }}</label>

                         <div class="col-md-6">
                            <input id="email" type="email" class="form-control
@error('email') is-invalid @enderror" name="email" value="{{ old('email') }}" required
autocomplete="email" autofocus>

                            @error('email')
                               <span class="invalid-feedback" role="alert">
                                  <strong>{{ $message }}</strong>
                               </span>
                            @enderror
                         </div>
                      </div>

                      <div class="form-group row">
                         <label for="password" class="col-md-4 col-form-label text-md-
right">{{ __('Password') }}</label>

                         <div class="col-md-6">
                            <input id="password" type="password" class="form-control
@error('password') is-invalid @enderror" name="password" required
autocomplete="current-password">
```

```
                        @error('password')
                            <span class="invalid-feedback" role="alert">
                                <strong>{{ $message }}</strong>
                            </span>
                        @enderror
                    </div>
                </div>

                <div class="form-group row">
                    <div class="col-md-6 offset-md-4">
                        <div class="form-check">
                            <input class="form-check-input" type="checkbox"
name="remember" id="remember" {{ old('remember') ? 'checked' : '' }}>

                            <label class="form-check-label" for="remember">
                                {{ __('Remember Me') }}
                            </label>
                        </div>
                    </div>
                </div>

                <div class="form-group row mb-0">
                    <div class="col-md-8 offset-md-4">
                        <button type="submit" class="btn btn-primary">
                            {{ __('Login') }}
                        </button>

                    </div>
                </div>
            </form>
        </div>
    </div>
</div>
</div>
@endsection
```

Αρχείο app → resources → views → auth → passwords → confirm.blade.php

```
@extends('layouts.app')
```

```
@section('content')
<div class="container">
    <div class="row justify-content-center">
        <div class="col-md-8">
            <div class="card">
                <div class="card-header">{{ __('Confirm Password') }}</div>

                <div class="card-body">
                    {{ __('Please confirm your password before continuing.') }}

                    <form method="POST" action="{{ route('password.confirm') }}">
                        @csrf

                        <div class="form-group row">
                            <label for="password" class="col-md-4 col-form-label text-md-
right">{{ __('Password') }}</label>

                            <div class="col-md-6">
                                <input id="password" type="password" class="form-control
@error('password') is-invalid @enderror" name="password" required
autocomplete="current-password">

                                @error('password')
                                    <span class="invalid-feedback" role="alert">
                                        <strong>{{ $message }}</strong>
                                    </span>
                                @enderror
                            </div>
                        </div>

                        <div class="form-group row mb-0">
                            <div class="col-md-8 offset-md-4">
                                <button type="submit" class="btn btn-primary">
                                    {{ __('Confirm Password') }}
                                </button>

                                @if (Route::has('password.request'))
                                    <a class="btn btn-link" href="{{ route('password.request') }}">
                                        {{ __('Forgot Your Password?') }}
                                    </a>
                                @endif
                            </div>
                        </div>
```

```
                </form>
            </div>
        </div>
    </div>
</div>
@endsection
```