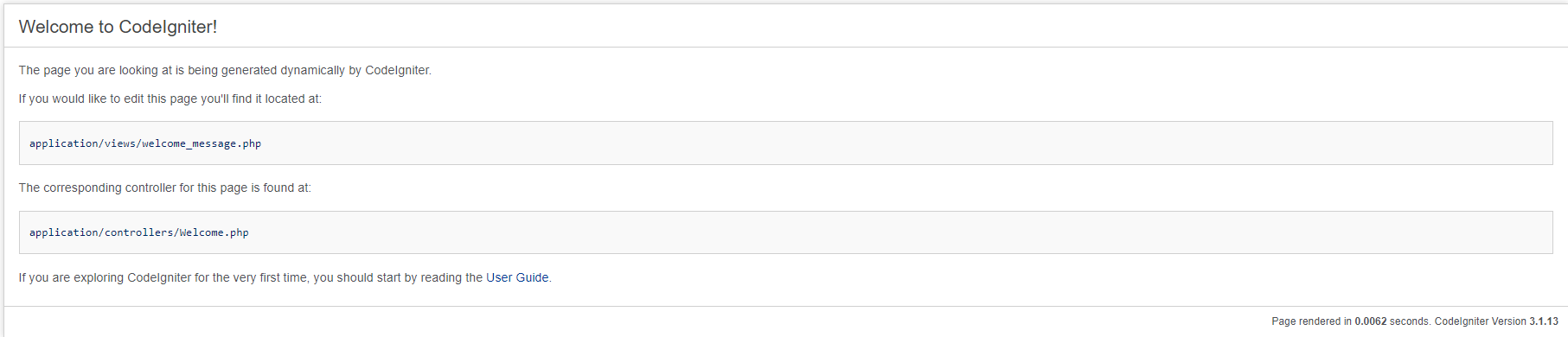
Αρχικά έκανα install το CodeIgniter3 το Wamp και τη HeidiSQL. Στο wamp πέρασα στο φάκελο www το codeigniter file που εκεί μπορείς να βάλεις όλα τα projects που θες να κάνεις Localhost.Το ονόμασα trytodo. Php Version : 8.0.26

Για αρχή για να ελένξω ότι το setup έχει γίνει σωστά, άνοιξα το wamp και στο browser μου έγραψα <http://localhost/trytodo> (όπου το trytodo είναι το όνομα του project) και έβαλα στο config το base\_url που είναι το http://localhost/trytodo



Αφού εμφάνισε τη συγκεκριμένη σελίδα σημαίνει ότι το CodeIgniter έχει γίνει σωστά install.

Έφτιαξα το register api για αρχή στο οποίο θα είναι το registration page. Χρησιμοποίησα έτοιμο template από bootstrap. Επειδή το api δε μου δούλευε με το wamp στην αρχή δοκίμασα να τρέξω το server με php –S localhost:8000 και δούλευε κανονικά οπότε χρησιμοποίησα το .htaccess αρχείο το οποίο κάνει enable το url rewriting.

Αρχικά έφτιαξα τα endpoints για το register και login τα controllers και views τους. Στο register έβαλα τους περιορισμούς όπως το email να είναι valid και παράλληλα να ελέγχει τη database για αν υπάρχει ήδη το email από άλλο user, τα passwords να είναι ίδια και να μην μπορεί να κάνει register αν δεν έχει συμπληρώσει όλα τα στοιχεία. Έφτιαξα την db με όνομα trytodo και ένα users table το οποίο κρατάει όλα τα στοιχεία του register και με ένα παραπάνω column role\_as με με default το 1 που είναι για απλούς users για να χρησιμοποιήσω αργότερα που θα ελέγχει αν ο user είναι απλός ή admin. Σύνδεσα την Heidi με το codeigniter στο database.php

Στο LoginController έβαλα να ελέγχει αν υπάρχουν τα στοιχεία που γράφει ο χρήστης στην βάση δεδομένων και μετά να τον μεταφέρει στο userpage το οποίο είναι το homepage του site(έφτιαξα και το api παράλληλα ). Σε περίπτωση που ο χρήστης έγραφε λάθος τα στοιχεία του εμφανίζει σχετικό μήνυμα και τον κάνει redirect στο loginpage ξανά.

Πήρα ως header ένα έτοιμο template και το άλλαξα για να ταιριάζει με αυτό του trytodo και να λειτουργούν τα κουμπια όταν τα πατάει ο χρήστης. Σε κάθε κουμπί έβαλα το ανάλογο endpoint. Για να μην εμφανίζει μόνιμα το header και πριν το login του χρήστη και στο register ξεκίνησα να χτίσω το authentication system το οποίο δεν αφήνει τον χρήστη να μπει απευθείας στο homepage χωρίς να κάνει login και εμφανίζει σχετικό μήνυμα. Επίσης έβαλα στο header αν δεν είναι authenticated χρήστης να μην το εμφανίζει οπότε αν δεν κάνει Login να μην υπάρχει το header.

Μετά έφτιαξα και ένα logout button για να μπορεί ο χρήστης να κάνει Logout και να μπορέσω επίσης να δοκιμάσω όλες τις λειτουργίες του authentication για το αν ξέχασα κάτι ή δε δουλεύει κάτι σωστά

Έπειτα πρόσθεσα όταν ο user κάνει register να του στέλνει στο email ένα verification link το οποίο πάει και στο database και αυτόματα ο κάθε user παίρνει στο db το is\_email\_verified = no και μόλις ανοίξει το link το μετατρέπει σε yes.

Ξεκίνησα έπειτα με το Update του Profile του χρήστη. Για αρχή έβαλα ένα form\_validation με το οποίο να μην μπορέι να πατήσει update αν δεν βάλει όλα τα στοιχεία έτσι ώστε να μην γίνει update σε κενά στο db τα στοιχεία του. Και επίσης με την μεταβλητή logged\_in η οποια είναι το authentication system που είχα φτιάξει στο LoginController την δέχεται στο UpdateController έτσι ώστε ο current logged\_in user να μπορεί να κάνει μόνο στο δικό του column Update το Profile του. Και έθεσα και τα σχετικά μηνύματα για το αν είναι επιτυχής ή όχι το update του.

Mετά έφτιαξα το API για τα μηνύματα που θα μπορεί να στέλνει ο user και το table messages στην sql με τα columns id(id του κάθε μηνύματος), user\_id(το id του user που έστειλε το μήνυμα), body\_text(περιεχόμενο μηνύματος) και created\_at(ημερομηνία και ώρα δημιουργίας). MessageModel με το function create για να μπούν τα δεδομένα στην βάση και με το MessageController function create να του επιτρέπεται να στείλει μήνυμα μόνο όταν είναι κάποιος logged\_in και το μήνυμα να μην είναι άδειο. Κάνει Load το view file της φόρμας του μηνύματος επίσης.

MessageHistory έφτιαξα στο model αρχικά να παίρνει τα δεδομένα της sql και να δείχνει τα μηνύματα μόνο του συγκεκριμένου logged\_in user δηλαδή με το συγκεκριμένο του user\_id έτσι ώστε να μπορεί να δει τα δικά του μηνύματα μόνο. Κάνει load το history view file. Το οποίο εμφανίζει τα μηνύματα με σειρά αποστολής και έχει τη λειτουργία του expand και collapse.

Έπειτα έκανα copy paste το header σε ένα ίδιο με το όνομα headeradmin το οποίο θα εμφανίζεται μόνο σε όσους έχουν το admin ρόλο(AdminController), δηλαδή στο role\_as = 2 στη βάση.

Admin Account: [admin@gmail.com](mailto:admin@gmail.com) Password: 1

Στο AdminController έφτιαξα να μπορεί και ο admin να αλλάξει τα στοιχεία του. Στη στήλη customers να μπορεί να δει τους registered users και πατώντας στον καθένα τους να μπορεί να τους κάνει delete edit ή να δει τα μηνύματα του καθενός. Για το delete έθεσα να παίρνει το συγκεκριμένο user\_id του selected user και όταν πατάει το delete να τον διαγράφει από την βάση. Για το edit πάλι έπαιρνα τον selected user και η διαδικασία ήταν σαν το update που καταχώρουσα τα νέα στοιχεια στη βάση δεδομένων και ξαναέκανα hash το νέο password. Για το Message History του selected user με το function get\_messages πήρα τα μηνύματα και σύγκρινα το user\_id με το id του user από τα 2 tables και αν ήταν το ίδιο έδειχνε τα μηνύματα του χρήστη στον admin. Τέλος για το MessageHistory του admin απλά έβαλα τα messages όλων των users να υπάρχουν μαζί και με το id αυτού που το έστειλε και ημερομηνία και ώρα.