

Project Report

Εφαρμογή Διαδικτύου Ενοικίασης Δωματίων / Κατοικιών

Ανδρέας Παππάς, 1115201500201

Spring Semester, 2019-20

Πίνακας Περιεχομένων

[Introduction](#)

[Κεφάλαιο 1ο](#)

[Σημαντικό!!!:](#)

[Κεφάλαιο 2ο](#)

[2.1](#)

[2.2](#)

[2.3](#)

[2.4](#)

[2.5](#)

[2.6](#)

[2.7](#)

[2.7.1](#)

[2.8](#)

[2.9](#)

[Κεφάλαιο 3ο](#)

[Git Logs](#)

[Initial Commit!](#)

[Create a simple Rails Project with Bootstrap](#)

[Create basic authentication](#)

[Building navbar with partial view](#)

[Authentication with full name](#)


[Update authentication views](#)

[Gravatar](#)

[Notification](#)

[Sending transactional email with Gmail](#)

[Sending transactional email with Mailgun](#)



[Create Facebook app](#)
[Create social authentication](#)
[Styling views](#)
[Create user profile page](#)
[Create edit profile page](#)
[Create Room Model](#)
[Create Room Controller](#)
[Create Room Views](#)
[Styling Views](#)
[Install paperclip](#)
[Create Photo Model](#)
[Create Photo Controller](#)
[Create Photo View](#)
[Remove Photos with AJAX](#)
[Amazon S3](#)
[Add Check to Room Views](#)
[Update the Photo Removing with AJAX](#)
[Issue with hidden fields](#)
[Create Room Index Page](#)
[Create Room Show Page](#)
[Add Google Map](#)
[Add Nearby Rooms](#)
[Create Reservations Model](#)
[Create Reservations View](#)
[Create Reservations Controller](#)
[Add jQuery Date Picker](#)
[Refactoring Reservation Form](#)
[AJAX for start date](#)
[AJAX for end date](#)
[Create Your Trips Page](#)
[Create Your Reservations Page](#)
[Modify User Profile Page](#)
[Creating Reviews Model](#)
[Creating Reviews Controller](#)
[Creating Reviews View](#)
[Create Show Reviews Page](#)
[Adding jquery raty](#)
[Add stars to reviews](#)

[Update Home Page](#)
[Creating Search Page](#)
[Create search function](#)
[Add Google Map](#)
[AJAX Searching](#)
[Add jquery pricing slider](#)
[Modify Home Page](#)
[Improving Home Page](#)
[Auto Location Suggestion](#)
[Dashboard Controller](#)
[Dashboard View](#)
[Instant/Request Booking Model](#)
[Instant/Request Booking Function](#)
[Approve/Decline Reservations](#)
[Reservation status](#)
[Calendar Controller](#)
[Host Calendar Page](#)
[Improving Host Calendar](#)
[Calendar Next/Back](#)
[Calendar availability model](#)
[Calendar Availability Form](#)
[Calendar Pricing](#)
[Sending Email for a Successful Booking \(could not get it to work\)](#)
[Conversations and Messages Model](#)
[Conversations and Messages Controller](#)
[Conversations and Messages View](#)
[Action Cable Configuration](#)
[Real time messages](#)
[Notification Model](#)
[Notification Controller](#)
[Notification View](#)
[Improving Notification View](#)
[Update Room Searching Function](#)
[Unavailable Dates on Date Pickers](#)
[Last Commit! Done!](#)

[Conclusion](#)

—

Introduction

1. Εξώφυλλο εργασίας
2. Εισαγωγή
3. Κεφάλαια ανάλυσης εργασίας
4. Επίλογος
5. Πίνακας περιεχομένων & git logs

Στόχος της εργασίας: Στόχος της εργασίας ήταν η υλοποίηση / απομίμηση της πλέον πασίγνωστης εφαρμογής παγκόσμιου ιστού ονόματι: *airbnb*.

Περιεχόμενο των κεφαλαίων που ακολουθούν: Παρακάτω θα αναφερθεί με βηματική σειρά, (από την αρχή εκπόνησης της εργασίας μέχρι το τέλος της) ο τρόπος με τον οποίο έχει υλοποιηθεί η εργασία, οι τεχνολογίες που χρησιμοποιήθηκαν, καθώς και τα git logs τα οποία επιδεικνύουν βήμα βήμα την δουλειά που έχει γίνει.

Τυχόν παραδοχές θα αναφέρονται μέσα στα κεφάλαια όπου αυτές βρίσκονται.

Τρόπος χρήσης εφαρμογής (prerequisites): Θα πρέπει να υπάρχουν εγκατεστημένα στον υπολογιστή μας: (αναφορά για: σύστημα linux)

1. Η γλώσσα προγραμματισμού ruby, ruby on rails ([info](#)), επειδή η ruby είναι πολύ αυστηρή όσον αφορά τα versions της, για να λειτουργήσει σωστά η εφαρμογή και να αποφύγουμε τυχόν errors θέλουμε τις εκδόσεις: (ruby: 2.5 & rails: 5.0.3)
2. Η βάση δεδομένων sqlite3 & το εργαλείο DB Browser for sqlite3 ([info](#))
3. Εφόσον έχουμε εγκαταστήσει τα παραπάνω, ανοίγουμε τερματικό και πηγαίνουμε στον κατάλογο της εργασίας και το πρώτο πράγμα που πληκτρολογούμε είναι: **bundle install** (θα πρέπει να έχει εγκατασταθεί το bundler gem από το βήμα 1)
4. Μετά πληκτρολογούμε **rails db:migrate** (θα πρέπει να έχουμε ολοκληρώσει το βήμα 2)
5. Και τέλος πληκτρολογούμε **rails s** για να ξεκινήσουμε τον σερβερ και ανοίγοντας τον browser μας πληκτρολογούμε: **http://localhost:3000**

Κεφάλαιο 1ο

Αρχικά να ξεκινήσω δίνοντας τα credentials για το **admin** page:

- email: andrewpap@outlook.com (admin)
- password: password

Δίνοντας τα ανωτέρω στο login page πηγαίνουμε αυτόματα στο admin dashboard.

Σημαντικό!!!:

Επειδή το mailgun account όπου χρησιμοποιώ στην εργασία για να στέλνω τα confirmation emails για το registration έληξε το free trial period και πρέπει πλέον να κάνω πρώτα add το email address στο mailgun account μου του οποίου είναι να του στείλουμε confirmation email for approval, δίνω παρακάτω άλλα 2 accounts που είναι ήδη εγγεγραμμένα στην βάση για λόγους testing της λειτουργικότητας της εργασίας:

- email: andrewpap1997@gmail.com (guest)
- password: qwerty22
- email: darktomorrowland@gmail.com (host)
- password: test22

Να σημειωθεί επίσης πως έχω υλοποιήσει την ίδια λειτουργία του mailgun με το gmail API, αλλά έχω αφαιρέσει από την εφαρμογή τα προσωπικά μου credentials, οπότε η επίδειξη ορθής λειτουργίας του αν αυτή κριθεί σκόπιμη θα παρουσιαστεί στην εξέταση της εργασίας!

Επίσης το extra functionality login with facebook, θα λειτουργήσει μόνο αν κάνετε provide τα δικά σας API keys, fb_app_id & fb_app_secret καθώς τα δικά μου τα έχω αφαιρέσει από την εργασία. (η επίδειξη λειτουργικότητας μπορεί να γίνει στην εξέταση της εργασίας)

Τώρα μπορούμε να προχωρήσουμε στο κεφάλαιο 2 όπου και θα ξεκινήσει η ανάλυση υλοποίησης της εργασίας!

Κεφάλαιο 2ο

2.1

Αρχικά να ξεκινήσω λέγοντας πως η εφαρμογή μου υλοποιεί όλα όσα ζητάει η εκφώνηση με αρκετά extra functionalities, τα οποία θα αναφερθούν παρακάτω αναλυτικά.

Ξεκίνησα εγκαθιστώντας τα κατάλληλα ruby gems, (bootstrap, devise gem) και ξεκίνησα να χτίζω το navigation bar το οποίο πήρα αρχικά από τα bootstrap [examples](#) και το μορφοποίησα ανάλογα με plain CSS. Μετά έστησα το devise authentication να είναι με full name, εκτελώντας: generate migration AddFullnameToUser fullname:string η οποία εντολή χρησιμοποιείται πάρα πολλές φορές κατά την υλοποίηση της εργασίας, και απλά προσθέτει τα πεδία τα οποία επιθυμούμε στην βάση δεδομένων μας με τις ανάλογες πληροφορίες που θέλουμε να προσθέσουμε. Να σημειωθεί εδώ πως το devise gem είναι υπεύθυνο για το στήσιμο του log in / register και κάνει encrypt αυτόματα τα password την βάση μας. ([info](#))

Μετά μορφοποίησα με plain CSS τα authentication views και πρόσθεσα bootstrap στα devise forms. Επίσης στην μπάρα πλοήγησης βλέπουμε το εικονίδιο του χρήστη όπου κάνει successfully login το οποίο είναι υλοποιημένο απο: [gravatar](#). Στη συνέχεια υλοποίησα το [notification](#) χρησιμοποιώντας το toastr-rails gem & jQuery, το οποίο φαίνεται πάνω δεξιά όταν κάνουμε successfully register, login αλλά και σε error cases και μορφοποίησα με plain CSS. Μετά υλοποίησα την αποστολή confirmation emails στους καινούργιους λογαριασμούς που κάνουν register χρησιμοποιώντας 2 διαφορετικούς τρόπους:

1. [Mailgun API](#)
2. [Gmail API](#)

Βέβαια το gmail όπως είναι γνωστό δεν είναι και η καλύτερη επιλογή καθώς υπάρχουν πολλά [limitations](#) ενώ το mailgun ήταν μια πολύ καλύτερη επιλογή πλέον και αυτό έχει χρονικό όριο στο free trial period του, αλλά για το testing της εργασίας κάνει την δουλειά του.

Επόμενο βήμα το οποίο είναι **extra functionality** καθώς δεν ζητείται από την εκφώνηση είναι το registration & sign in μέσω facebook account. Η ανάλυση υλοποίησής του θα καλυφθεί στο κεφάλαιο 3 καθώς και σε ποιά αρχεία του project και τι πρέπει να αλλαχθεί για την λειτουργικότητά του σε άλλον υπολογιστή.

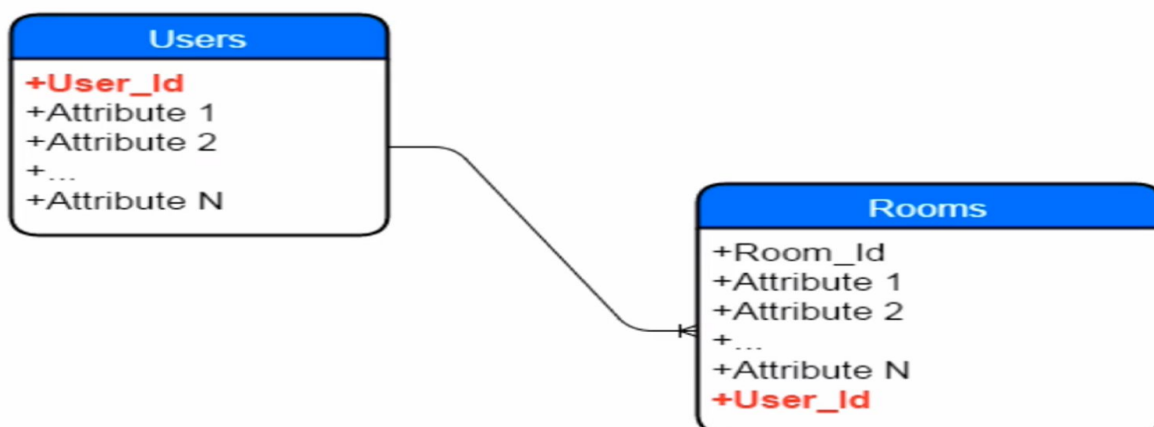
2.2

Στην συνέχεια σχεδιάσα την **σελίδα προφίλ για κάθε χρήστη** (το οποίο επίσης δεν ζητείται από την εκφώνηση). Πρόσθεσα και μερικά έξτρα πεδία στην βάση δεδομένων για κάθε χρήστη. Μετά σχεδιάσα την σελίδα επεξεργασίας προφίλ για κάθε χρήστη όπου εκεί μπορεί να αλλάξει το ονοματεπώνυμό του, να προσθέσει τηλέφωνο επικοινωνίας, να γράψει κάποιο description για τον ίδιο, να αλλάξει email & password αν θέλει καθώς και υπάρχει κουμπί προβολής προφίλ στην ίδια σελίδα όπου τον στέλνει στο προφίλ του και το πως το βλέπουν οι άλλοι χρήστες. Στην σελίδα προφίλ εμφανίζεται η εικόνα προφίλ κάθε χρήστη την οποία λαμβάνουμε από το gravatar, αν είναι verified χρήστης από email ή facebook account, ονοματεπώνυμο, ότι πληροφορίες έχει εισάγει ο ίδιος καθώς και τα σπίτια τα οποία έχει διαθέσιμα προς ενοικίαση αν είναι οικοδεσπότης καθώς και κριτικές από άλλους χρήστες (οικοδεσπότες ή ενοικιαστές).

Μία **παραδοχή** την οποία ξέχασα να αναφέρω παραπάνω είναι η εξής: *στην εργασία ζητείται κατά την εγγραφή, ο χρήστης να επιλέξει αν θα είναι οικοδεσπότης ή ενοικιαστής. Επειδή με το devise gem δεν μου ήταν εφικτό να υλοποιήσω κάτι τέτοιο, αυτό που έχω κάνει είναι να στέλνω email επιβεβαίωσης σε οποιονδήποτε χρήστη κάνει εγγραφή και καθώς επιβεβαιώνεται η διεύθυνσή του και εισέρχεται στην εφαρμογή και στην μπάρα πλοήγησης υπάρχει το κουμπί: [become a host] το οποίο στέλνει τον χρήστη στην σελίδα list your space την οποία μπορούμε να βρούμε και από το dropdown menu και εκεί μπορεί να δημιουργήσει το σπίτι (καθώς και όλες τις πληροφορίες για αυτό) το οποίο είναι προς ενοικίαση και με αυτόν τον τρόπο γίνεται host. Και αν δεν το κάνει αυτό παραμένει guest. Ο host μπορεί να είναι ταυτόχρονα και guest με τον ίδιο λογαριασμό. Προφανώς σε αυτήν την περίπτωση ενημερώνω την βάση πως ο συγκεκριμένος χρήστης με συγκεκριμένο used_id πλέον είναι host και έχει επιπλέον functionalities από τον guest & φυσικά τον ανώνυμο χρήστη. **Πέρα από αυτό το σημείο δεν υπάρχει άλλη παραδοχή στην εργασία και όλα είναι υλοποιημένα ακριβώς όπως τα ζητάει η εκφώνηση χωρίς να λείπει κάτι, αντιθέτως υπάρχουν επιπλέον προσθήκες όπως ήδη έχουμε αναφέρει.***

2.3

Στην συνέχεια δημιουργώ στην βάση το μοντέλο των δωματίων. [*Rails g model Room with many fields*] Τα fields όπου δίνουμε για κάθε δωμάτιο είναι όσα ζητάει η εκφώνηση.



Στην ανωτέρω εικόνα βλέπουμε την σχέση την οποία έχουμε στην βάση μας μεταξύ χρηστών και δωματίων. Ένας χρήστης στην περίπτωση που είναι host μπορεί να έχει στην κατοχή του 1 ή περισσότερα δωμάτια προς ενοικίαση για αυτόν τον λόγο δίνουμε το foreign key User_id στο μοντέλο των δωματίων το οποίο προφανώς είναι primary key στους users και προφανώς τα δωμάτια έχουν τα πολλαπλά attributes που ζητάει η εκφώνηση. Επίσης ένας χρήστης αν είναι guest μπορεί να νοικιάσει 1 ή πολλαπλά δωμάτια οπότε εξίσου χρειάζεται η ανωτέρω σχέση. Στην παρακάτω εικόνα παραθέτω ένα παράδειγμα της παραπάνω σχέσης:

ID	Name		ID	Description	Location	User_ID
1	John	→	1	Nice house	New York	1
2	David	→	2	Quite place	San Francisco	1
3	Ben	→	3	Modern room	Melbourne	2

Όπως βλέπουμε ο John έχει κάποια σχέση και με το σπίτι 1 (Νέα Υόρκη) αλλά και με το 2 (Σαν Φρανσίσκο) καθώς βλέπουμε το user_id του (1) να εμφανίζεται στα 2 πρώτα αυτά σπίτια, και είναι αυτό που επιδεικνύουν τα βελάκια. Ο Ben από την άλλη έχει κάποια σύνδεση μόνο με το σπίτι νούμερο 3 (Μελβούρνη).

2.4

Στην συνέχεια, αφού έχουμε ολοκληρώσει το μοντέλο των δωματίων, δημιουργούμε το room controller όπου υλοποιώ τις ανωτέρω θεωρητικές σχέσεις που αναφέραμε, πρακτικά στην εφαρμογή και δημιουργώ τις διάφορες σελίδες πλοήγησης. (Σε ποιους φακέλους και αρχεία έχει γίνει τι ακριβώς θα φανεί στο κεφάλαιο 3 στα git logs) Μετά φτιάχνω τα views για την κάθε σελίδα και την μορφοποιώ με Sass & Bootstrap. Οι ανωτέρω σελίδες που δημιουργήθηκαν αφορούν στην δημιουργία και επεξεργασία δωματίων από τους hosts. (Όπως τα ζητάει η εκφώνηση, με αναλυτικά comments στον κώδικα και φαίνεται η λειτουργικότητα στην εφαρμογή)

Έπειτα κάνω install το [paperclip](#) το οποίο και χρησιμοποιώ για να ανεβάζει ο κάθε host 1 ή παραπάνω φωτογραφίες για κάθε δωμάτιο. Δημιουργώ το photo model, controller, view και ύστερα χρησιμοποιώ AJAX και κάποια εικονίδια στο frontend από το [fontawesome](#) για να μπορεί ο host, να διαγράψει όποια φωτογραφία θέλει και η αλλαγή αυτή να φαίνεται κατευθείαν χωρίς να χρειάζεται να κάνουμε refresh την σελίδα για να φανεί ότι διαγράφηκε όντως η φωτογραφία. Στην συνέχεια δημιουργώ amazon s3 bucket για το upload των φωτογραφιών αντί να γίνονται store τοπικά, αλλά πλέον χρειάζεται billed account με credit card, οπότε δεν μπόρεσα τελικά να το χρησιμοποιήσω...

Μετά πρόσθεσα στο sidebar της δημιουργίας των δωματίων ένα checkmark το οποίο θα εμφανίζεται κάθε φορά που ο οικοδεσπότης έχει ολοκληρώσει τις αλλαγές για κάθε δωμάτιο και αφού έχει ολοκληρώσει όλες τις απαραίτητες αλλαγές/ προσθήκες για το δωμάτιό του θα εμφανίζεται στο κάτω μέρος της σελίδας ένα publish button, όπου πιέζοντάς το θα δημοσιεύεται στην εφαρμογή το σπίτι του το οποίο είναι πλέον διαθέσιμο προς ενοικίαση.

Σε αυτό το σημείο, ένα πρόβλημα που αντιμετώπισα ήταν, πως τα html inputs μου στην φόρμα αλλαγών των δωματίων αρχικά τα είχα ως τύπου hidden. Το πρόβλημα με αυτό είναι πως, κάποιος θα μπορούσε να πάει στο google chrome developer tools και να αλλάξει on the fly το value των inputs όπου έκανα τον έλεγχο αν έχουν ολοκληρωθεί όλες οι αλλαγές να εμφανίζεται το publish button από true σε false. Αυτό θα άλλαζε την τιμή και μέσα στην βάση δεδομένων μας του input με αποτέλεσμα και αφού να έχουν ολοκληρωθεί όλες οι αλλαγές / προσθήκες δωματίου, να μην εμφανίζεται τελικά το publish button. Πως επιλύθηκε το παραπάνω? Δεν θα πρέπει να στηριζόμαστε μόνο στα hidden fields της html, γιατί όπως είδαμε κάποιος πιο 'έμπειρος' χρήστης, μπορεί να πειράξει την εφαρμογή μας όπως θέλει, οπότε θα πρέπει πάντα να ελέγχεται το logic στο server side. Οπότε πολύ απλά πηγαίνουμε στον controller των rooms και δημιουργούμε μια μέθοδο η οποία ελέγχει αν έχουν ολοκληρωθεί οι αλλαγές / προσθήκες και αν ναι, τότε εμφανίζεται το κουμπί, αλλιώς όχι. Και με αυτόν τον τρόπο δεν έχει σημασία τι θα πειράξει ο κακόβουλος χρήστης στο frontend μέσω της κονσόλας.

2.5

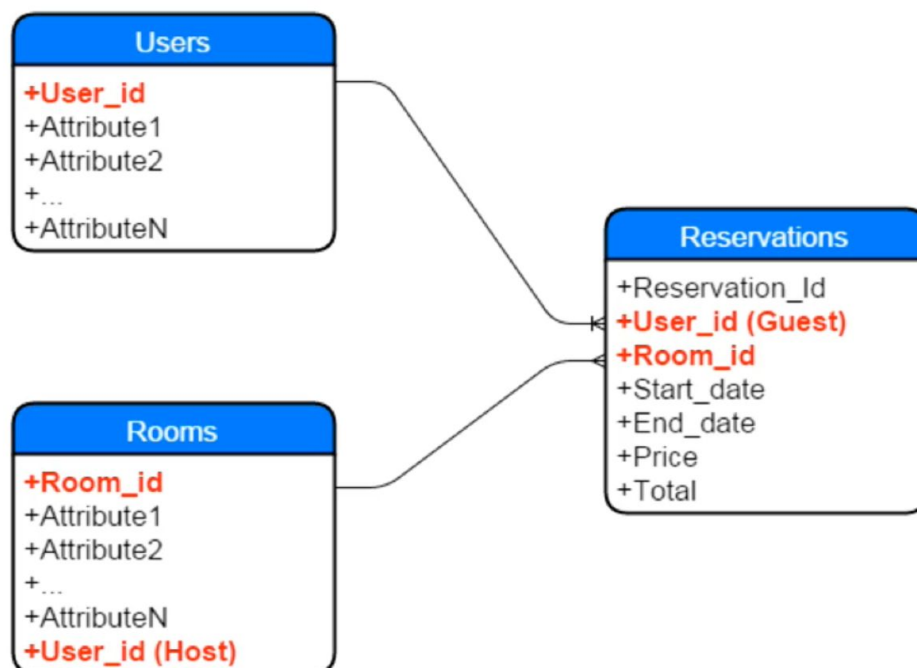
Έπειτα έχουμε την δημιουργία σελίδας προβολής δωματίων (ώστε να μπορεί να βλέπει ο χρήστης αναλυτικά πληροφορίες για κάθε δωμάτιο) η οποία ακολουθεί τις προδιαγραφές της εκφώνησης με την έξτρα υλοποίηση, στο κάτω κάτω μέρος της σελίδας να εμφανίζονται τα δωμάτια τα οποία είναι διαθέσιμα και κοντινά στο συγκεκριμένο δωμάτιο όπου βλέπουμε, καθώς και την απόστασή τους σε χιλιόμετρα και link πάνω στο όνομά τους ώστε να δούμε αναλυτικές πληροφορίες και για αυτά. Στην εφαρμογή για την προβολή του χάρτη και των δωματίων στους χάρτες χρησιμοποίησα google map & [geocoder gem](#), το οποίο όταν ο host πληκτρολογεί την διεύθυνση του δωματίου, αν είναι valid address που βρίσκεται στην βάση δεδομένων του google maps, θα πάρει την διεύθυνση αυτή και θα την μετατρέψει σε συντεταγμένες τις οποίες και δίνουμε στο google maps και με αυτόν τον τρόπο εμφανίζουμε την τοποθεσία στον χάρτη. Επίσης εμφανίζω και το βελάκι πάνω στην τοποθεσία όπου βρίσκεται το δωμάτιο και μια μικρή εικόνα του δωματίου. (την πρώτη από όσες περιλαμβάνει το δωμάτιο) Το πρόβλημα βέβαια με το [google maps api](#) πλέον είναι πως έγινε και αυτό billed και θα πρέπει να συμπεριλάβεις credit card για να μπορείς να το χρησιμοποιήσεις στις εφαρμογές σου. Οπότε αν πάμε να γράψουμε μια valid address σε κάποιο νέο listing, δεν θα ενημερωθεί αυτόματα η βάση μας με τα coordinates, οπότε θα πρέπει να πάμε να τα προσθέσουμε μόνοι μας άμα θέλουμε να δούμε τον χάρτη στην προβολή δωματίου αλλά και στα αποτελέσματα του searching όπως θα δούμε παρακάτω. Βέβαια η εφαρμογή είναι υλοποιημένη ορθά οπότε αν έχετε κάποιο δικό σας google api key μπορείτε να το προσθέσετε στα κατάλληλα αρχεία όπου είναι υλοποιημένο το google maps και η παραπάνω λειτουργία που περιγράψαμε θα λειτουργήσει κανονικά. Για λόγους testing έχω προσθέσει μόνος μου κάποια random coordinates τα οποία λειτουργούν κανονικά και βλέπουμε τον χάρτη αλλά και την ορθή τοποθεσία όπου αντιπροσωπεύουν οι συντεταγμένες (οπότε και αν δώσουμε στο searching όπως θα δούμε αργότερα την ορθή διεύθυνση που αντιστοιχεί στις συντεταγμένες θα δούμε τον χάρτη να εμφανίζεται ορθά). Όσο για το ποια αρχεία στην εφαρμογή αφορούν τα παραπάνω θα το δούμε στο κεφάλαιο 3 στα git logs.

Μετά δημιουργώ το μοντέλο των κρατήσεων (reservations), για να μπορεί ο χρήστης να κάνει κράτηση δωματίου στην σελίδα των δωματίων. Αρχικά οι κρατήσεις είναι instant και αργότερα προσθέτω την δυνατότητα να κάνει request για κράτηση και να βλέπει το αίτημα ο οικοδεσπότης και να μπορεί να απορρίπτει ή να δέχεται το αίτημα κράτησης από τον χρήστη. [Παρακάτω θα δούμε πως ακριβώς λειτουργεί το μοντέλο των κρατήσεων στην βάση]

Μπορεί να επιλεγθεί η ημερομηνία προσέλευσης καθώς και αποχώρησης μέσω ημερολογίου όπου εμφανίζεται όταν κλικάρουμε στην φόρμα ([jQuery date picker](#), AJAX for start, end date).

[Παρακάτω θα δείξουμε την ακριβή λειτουργία του AJAX για τις ημερομηνίες]

Users (as Guest) - Rooms

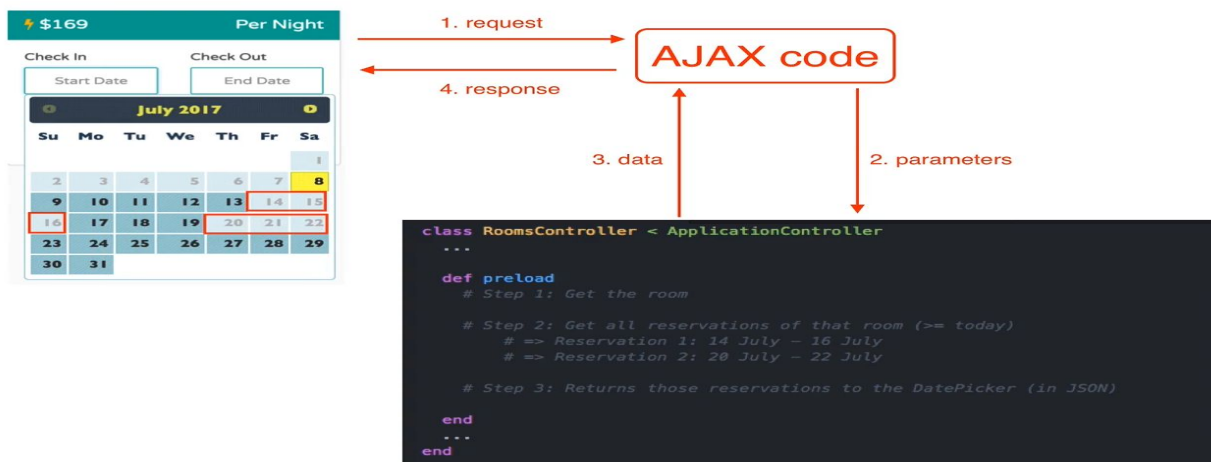


Παραπάνω στο κεφάλαιο 2 (παράγραφος 2.3), είχαμε μιλήσει για την σχέση των χρηστών και των δωματίων στην βάση δεδομένων μας. Εδώ λοιπόν μπορούμε να δούμε πως ολοκληρώνεται η σχέση αυτή στην εφαρμογή μας, καθώς οι ενοικιαστές συμπεριλαμβάνονται στο μοντέλο των κρατήσεων το οποίο συνδέεται με το μοντέλο των δωματίων και το μοντέλο των οικοδεσποτών τελικά. **Ένα ή περισσότερα δωμάτια κάποιου οικοδεσπότη μπορούν να έχουν 1 ή παραπάνω κρατήσεις, από έναν ή παραπάνω ενοικιαστές.** Αυτό ακριβώς περιγράφει και η εικόνα μας παραπάνω. Και έτσι ολοκληρώνεται η σχέση αυτή στην εφαρμογή μας.

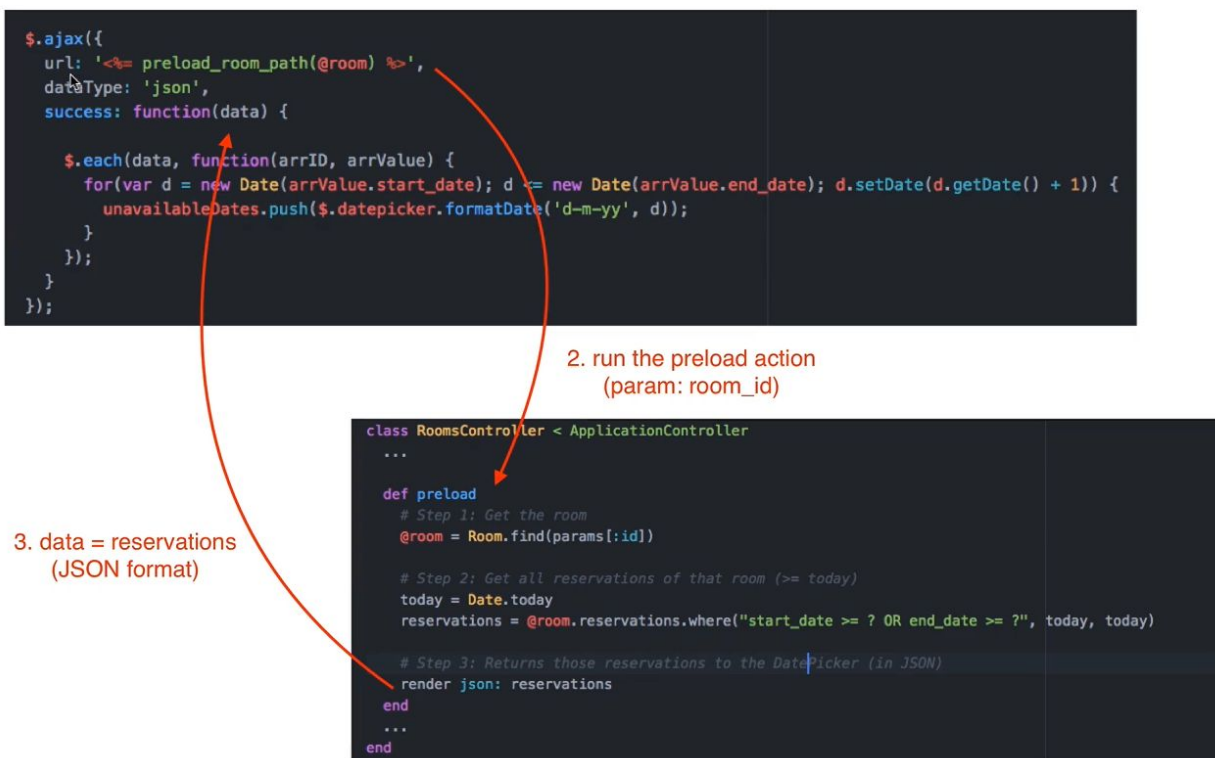
Στην παρακάτω εικόνα βλέπουμε την σχέση επικοινωνίας που έχει το frontend μας, και το backend μας, μέσω AJAX κώδικα. Ως πρώτο βήμα είναι το request που γίνεται στο backend αφού έχει επιλεγθεί κάποια ημερομηνία (στον ajax κώδικα). Βήμα 2ο είναι η κλήση της preload μεθόδου όπου εκεί παίρνουμε πρώτα το id του συγκεκριμένου δωματίου για το οποίο γίνεται η κράτηση, έπειτα παίρνουμε όλες τις κρατήσεις για το συγκεκριμένο δωμάτιο με ημερομηνία

μεγαλύτερη της σημερινής (όπου προφανώς η σημερινή είναι η τωρινή ημερομηνία όπου ο χρήστης κάνει την κράτηση) και επιστρέφουμε τις ημερομηνίες αυτές στον AJAX όπου από εκεί γίνεται το response και η επιστροφή σε json μορφή των ημερομηνιών στο ημερολόγιο.

Η ανωτέρω λειτουργία που περιγράψαμε φαίνεται γραφικά στην παρακάτω εικόνα:



Παρακάτω επίσης μπορούμε να δούμε τον κώδικα AJAX: (υλοποιεί τα όσα συζητήσαμε)

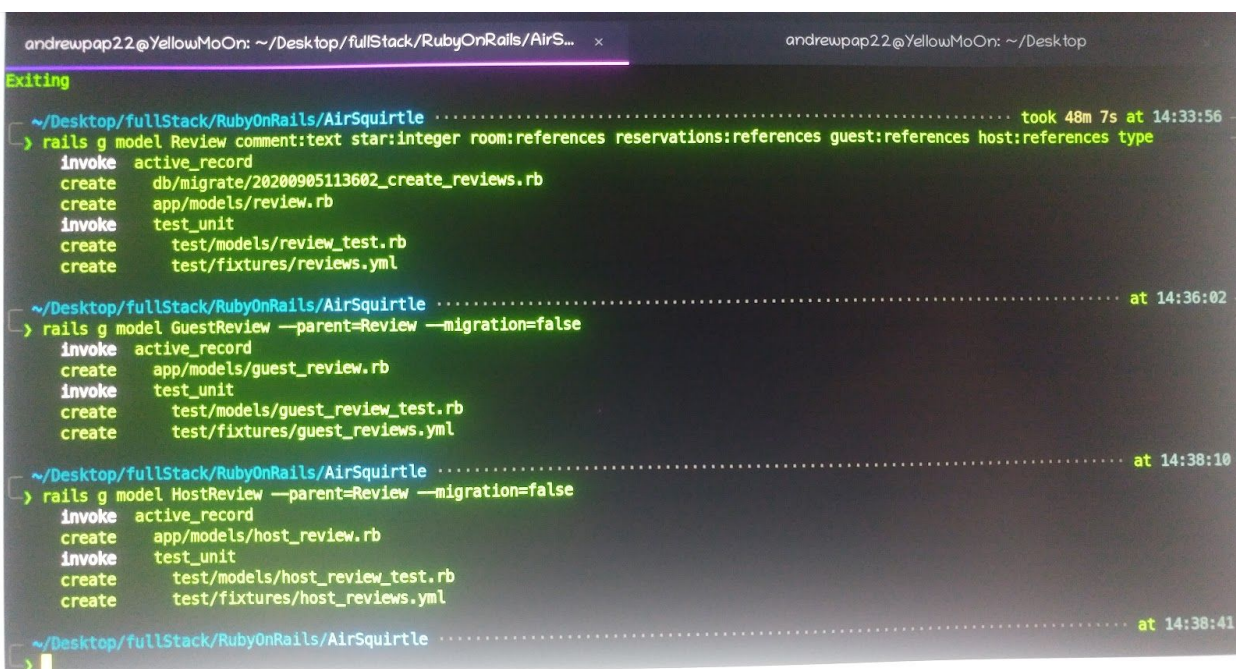


2.7

2 έξτρα προσθήκες τις οποίες δεν ζητάει η εκφώνηση είναι, η σελίδα **trips page & reservations page**. Όπου στην 1 μπορεί ο ενοικιαστής να δει όλα τα ταξίδια που έχει κάνει με βάση τις κρατήσεις των δωματίων που έχει κάνει και στην άλλη ο οικοδεσπότης μπορεί να δει τις κρατήσεις οι οποίες έχουν γίνει ή εκκρεμούν από άλλους χρήστες για όλα του τα διαθέσιμα σπίτια, ποιές ημερομηνίες αφορούν οι κρατήσεις αυτές, από ποιους χρήστες έχουν γίνει, καθώς και να αποδεχθεί ή να απορρίψει τις αιτήσεις των κρατήσεων από την σελίδα αυτή. Από την στιγμή που γίνει αποδεκτή κάποια κράτηση εμφανίζεται δίπλα από το όνομα του ενοικιαστή ένα review button όπου μπορεί ο οικοδεσπότης να γράψει κάποια κριτική για τον συγκεκριμένο ενοικιαστή. (περισσότερα για τα reviews παρακάτω)

Και έτσι φτάνουμε στο μοντέλο των κριτικών.

Ένα παράδειγμα της δημιουργίας των κριτικών στην βάση μας μέσω χρήσης rails φαίνεται στην παρακάτω εικόνα, καθώς και με τον ίδιο τρόπο δημιουργούμε όλα τα μοντέλα μας στην εφαρμογή.



```
andrewpap22@YellowMoOn: ~/Desktop/fullStack/RubyOnRails/AirS... x andrewpap22@YellowMoOn: ~/Desktop
Exiting

~/Desktop/fullStack/RubyOnRails/AirSquirtle ..... took 48m 7s at 14:33:56
> rails g model Review comment:text star:integer room:references reservations:references guest:references host:references type
  invoke  active_record
  create  db/migrate/20200905113602_create_reviews.rb
  create  app/models/review.rb
  invoke  test_unit
  create  test/models/review_test.rb
  create  test/fixtures/reviews.yml

~/Desktop/fullStack/RubyOnRails/AirSquirtle ..... at 14:36:02
> rails g model GuestReview --parent=Review --migration=false
  invoke  active_record
  create  app/models/guest_review.rb
  invoke  test_unit
  create  test/models/guest_review_test.rb
  create  test/fixtures/guest_reviews.yml

~/Desktop/fullStack/RubyOnRails/AirSquirtle ..... at 14:38:10
> rails g model HostReview --parent=Review --migration=false
  invoke  active_record
  create  app/models/host_review.rb
  invoke  test_unit
  create  test/models/host_review_test.rb
  create  test/fixtures/host_reviews.yml

~/Desktop/fullStack/RubyOnRails/AirSquirtle ..... at 14:38:41
>
```

Από την παραπάνω εικόνα γίνεται και κάπως εμφανής η σχέση των μοντέλων μας στην βάση για τις κριτικές. (περισσότερα παρακάτω)

2.7.1

Επεξήγηση του μοντέλου κριτικών:

Ένας χρήστης μπορεί να κάνει 1 κριτική για 1 κράτηση. Οπότε, για να αναγνωρίσουμε που ανήκει η κριτική αυτή θα πρέπει να γνωρίζουμε το id του χρήστη και της κράτησης. Για αυτό τον λόγο κάνουμε αναφορά & σύνδεση στην βάση μας στο μοντέλο των δωματίων και των χρηστών. Όμως αν προσέξουμε την παραπάνω εικόνα δεν έχουμε reference σε users αλλά σε guests & hosts.

Κανονικά δεν θα υπήρχε λόγος όπως έχουμε δει παραπάνω να προσθέσουμε reference στο room, γιατί γνωρίζοντας το id της κράτησης αυτομάτως γνωρίζουμε και για ποιο δωμάτιο μιλάμε, αλλά, με την προσθήκη αυτή μπορούμε να κάνουμε query πολύ εύκολα για να πάρουμε όλες τις κριτικές τις οποίες έχει ένα δωμάτιο από το να υλοποιήσουμε κάποιο πιο complex query μέσω του μοντέλου των κρατήσεων για να πάρουμε πίσω την ίδια πληροφορία.

Τώρα ο λόγος για τον οποίο έχουμε στην βάση μας reference σε hosts & guests αντί για users γενικώς είναι, διότι, οι χρήστες μας στην εφαρμογή μας όπως είπαμε μπορεί να είναι και hosts αλλά και guests ταυτοχρόνως, οπότε με αυτήν την παραλλαγή είναι πολύ εύκολο με την ανωτέρω προσθήκη στην βάση να υλοποιήσουμε ένα 2-way review system, όπου ενοικιαστές θα μπορούν να γράφουν κριτικές για οικοδεσπότες και το ανάποδο. Οπότε έτσι δημιουργούμε τα 2 μοντέλα όπως φαίνεται στην εικόνα: GuestReview & HostReview για να υπηρετήσουμε των ανωτέρω σκοπό. Όπως βλέπουμε όμως από το migration = false, αυτά τα 2 μοντέλα δεν θα είναι 2 νέα μοντέλα στην βάση αλλά θα λειτουργήσουν ως 'παιδιά' / αντικείμενα του μοντέλου των κριτικών για να ξεχωρίσουμε τα reviews των guests από των hosts. Άρα στην βάση μας τελικά άμα το πεδίο type του μοντέλου reviews είναι HostReview, αυτό σημαίνει πως ο οικοδεσπότης γράφει κριτική για τον ενοικιαστή, αλλιώς συμβαίνει το αντίθετο.

Το τελευταίο που μένει να εξηγήσουμε είναι, αφού δεν έχουμε guest & host μοντέλα αλλά απλά 1 users μοντέλο, πως αυτό αναλύεται σε guests & hosts. Πηγαίνοντας στην εφαρμογή μας στην διαδρομή: *app/models/guest_reviews.rb* θα δούμε την γραμμή κώδικα: *belongs_to :guest, class_name: "User"*. Αυτό που συμβαίνει εδώ είναι πως το guest λειτουργεί ως alias για το μοντέλο των users και έτσι ολοκληρώνεται η σύνδεση του μοντέλου των κριτικών & των χρηστών και με αυτόν τον τρόπο 'χωρίζουμε' τους users σε guests & hosts. (προφανώς το αντίστοιχο γίνεται στο αρχείο *host_reviews.rb* στον ίδιο κατάλογο)

Προφανώς εν τέλει ένας guest μπορεί να γράψει πολλές κριτικές για ένα δωμάτιο και έναν host, και ένας host μπορεί να γράψει πολλές κριτικές για έναν guest αλλά όχι προφανώς για δωμάτιό του! Και με την παραπάνω λειτουργία όπου ανέλυσα, όλα αυτά καθίστανται πολύ εύκολα.

2.8

Για την εμφάνιση των αστεριών στις κριτικές χρησιμοποιώ το jQuery [Raty](#).

Τέλος φτάνουμε στην αναβάθμιση και το στόλισμα του Home Page και την δημιουργία του Search Page όπως αυτή περιγράφεται στην εκφώνηση. Οποιοσδήποτε χρήστης από τους 4 ρόλους όπου αναφέρει η εκφώνηση, μπορεί να κάνει search στην αρχική σελίδα με βάση την τοποθεσία, διεύθυνση, αρχική & τελική ημερομηνία. Μετά θα εμφανιστούν κανονικά τα διαθέσιμα δωμάτια σε νέα σελίδα καθώς και ο χάρτης δεξιά των δωματίων με βελάκι πάνω στον χάρτη για το που βρίσκεται το καθένα και την τιμή τους να αναγράφεται πάνω στα βελάκια αυτά. Επίσης υπάρχει 'more filters button' το οποίο μέσω jQuery δημιουργούμε sliding interaction για να ανοίγει και να κλείνει κάθε φορά που το πατάει ο χρήστης. Μέσω αυτού ο χρήστης μπορεί να κάνει κάποια πιο σύνθετη αναζήτηση με όλες τις λειτουργίες που ζητάει η εκφώνηση και περισσότερες ακόμη.

Σημαντικό vol.2

Επειδή όπως αναφέραμε παραπάνω, στην παράγραφο 2.5 για το google maps api, για λόγους λειτουργικότητας και επίδειξης σας δίνω παρακάτω τις διευθύνσεις τις οποίες αν δώσετε στην εφαρμογή θα δείτε όλα τα αποτελέσματα και την πλήρη λειτουργικότητα της εφαρμογής. Ειδικά όμως άμα δώσετε κάποια διαφορετική από τις παρακάτω διευθύνσεις στο search bar, δεν θα δείτε κάποια διαθέσιμη κατοικία και η τοποθεσία στον χάρτη δεν θα είναι αντιπροσωπευτική της διεύθυνσης που δίνεται αλλά μια default που έχω προσθέσει στην βάση για λόγους λειτουργικότητας. Οι διευθύνσεις αυτές λοιπόν είναι:

1. **burwood, vic**
2. **rancho palos verdes**

Σε αυτό το σημείο να αναφέρω πως όταν ο χρήστης κάνει μια σύνθετη αναζήτηση με τις extra λειτουργικότητες, με χρήση AJAX δεν θα γίνεται reload ολόκληρη η σελίδα, αλλά μόνο ο χάρτης google maps και το footer της σελίδας όπου εμφανίζονται τα διαθέσιμα σπίτια. Καθώς η σελίδα αυτή είναι χωρισμένη στο frontend σε 3 διαφορετικά divs και έτσι η λειτουργικότητα αυτή για λόγους ταχύτητας είναι εύκολα υλοποιήσιμη!

Τελικά έκανα κάποιες διάφορες μορφοποιήσεις στην κεντρική σελίδα της εφαρμογής και ήθελα να προσθέσω την έξτρα λειτουργικότητα στο searchbar του [auto location suggestion](#) με χρήση geocomplete, κάτι το οποίο δεν κατάφερα λόγω του billed google maps api το οποίο δε μπορώ να χρησιμοποιήσω. Βέβαια ο κώδικας υπάρχει στην εφαρμογή, αλλά δεν είναι λειτουργικός.

2.9

Από αυτό το σημείο και έπειτα, έχω υλοποιήσει και θα αναφέρω όλες τις **extra λειτουργίες** της εργασίας μου που δε ζητάει η εκφώνηση καθώς και όλα τα improvements που έχω κάνει και τέλος θα μιλήσουμε για το admin page.

Υλοποίησα ένα dashboard page για τους χρήστες όπου μπορούν να δουν όλα τους τα σπίτια που έχουν κάνει list προς ενοικίαση καθώς και τυχόν notifications που μπορεί να έχουν από μηνύματα άλλων χρηστών. Επίσης έχω προσθέσει στην σελίδα αυτή ένα extra navigation bar που κατευθύνει στις σελίδες:

- Manage listings
- Messages
- Host Calendar

Το manage listings το έχουμε δει παραπάνω. Θα αναλύσουμε τα άλλα 2. Όσο για το πως έχουν υλοποιηθεί όλα αυτά θα φανούν στα git logs στο κεφάλαιο 3.

Υπάρχει όπως είπαμε η σελίδα Host Calendar από την οποία αν ο χρήστης έχει κάνει list ήδη κάποιο σπίτι, (δηλαδή είναι host) μπορεί να δει στο ημερολόγιο, για κάθε σπίτι του ξεχωριστά όπου επιλέγει για ποιο σπίτι θέλει να δει τι, απο το drop down menu στην αρχή της σελίδας, τις ημερομηνίες όπου έχουν κάνει (πράσινο χρώμα) ή ζητήσει (κίτρινο χρώμα) κράτηση, καθώς και από ποιόν χρήστη καθώς και να ορίσουν για ποιές ημερομηνίες είναι διαθέσιμο ή όχι το σπίτι προς ενοικίαση πατώντας πάνω στις αντίστοιχες ημερομηνίες, καθώς και να αλλάξουν την τιμή ενοικίασης για όποιες ημερομηνίες θέλουν!

Το messages page, δείχνει όλα τα μηνύματα που έχει ένας χρήστης από άλλους χρήστες καθώς και πριν πόση ώρα έχει σταλεί το μήνυμα, καθώς και αν έχει διαβαστεί ή όχι. Για να στείλει κάποιος χρήστης μήνυμα σε κάποιον άλλον αρκεί να πάει στο προφίλ του χρήστη πατώντας πάνω στο ονοματεπώνυμό του στην σελίδα δωματίου και μετά πατώντας το κουμπί send message (το οποίο κουμπί δεν εμφανίζεται σε μη εγγεγραμμένους χρήστες). (προφανώς αρχικά μπορούν να στείλουν μήνυμα μόνο οι guests σε hosts, αλλά μετά μέσω του notifications bar μπορούμε να δούμε από ποιόν έχουμε λάβει μήνυμα και να πάμε στο messages page και να ξεκινήσουμε διαδικασία 'chatting' η οποία είναι και real time με χρήση [action cable](#))

Όσο για τα improvements: πρόσθεσα τα unavailable dates να εμφανίζονται και στην σελίδα εμφάνισης δωματίων, όταν ένας χρήστης πάει να κάνει request ή instant booking, επίσης αναφορικά με αυτό, πρόσθεσα την δυνατότητα να επιλέγει ο οικοδεσπότης όταν καταχωρεί το σπίτι του στην εφαρμογή προς ενοικίαση, για να μπορεί ο guest να κάνει είτε request για κράτηση, είτε να μπορεί να κάνει απευθείας κράτηση. (εμφανίζεται ένας κεραυνός στο

ημερολόγιο στην 2η περίπτωση) Σε κάθε περίπτωση ο ιδιοκτήτης πρέπει να κάνει approve ή decline. Έπειτα εμφανίζεται η δυνατότητα για το 2 way review system.

*Μερικά ακόμη **έξτρα** που ήθελα να προσθέσω αλλά δεν μου έφτασε ο χρόνος είναι: Να κάνουμε verification κάθε χρήστη και με sms στο κινητό μέσω [twilio.com](https://www.twilio.com), να χρησιμοποιήσω [stripe](https://stripe.com) για κατάθεση πιστωτικής και online πληρωμή για τα δωμάτια, και να έφτιαχνα ένα *page & revenue chart* για να μπορεί ο host να βλέπει τα κέρδη του και σε στατιστική μορφή και την πορεία τους καθώς και να στέλνω sms, email επιβεβαίωσης σε *successful approved booking*! Ίσως τα υλοποιήσω στο μέλλον κάποια στιγμή!*

Και τέλος να κάνω αναφορά για το admin page το οποίο με δυσκόλεψε πάρα πολύ όπως αναφέρω και στον επίλογο για την υλοποίησή του from scratch αλλά τελικά μετά από πάρα πολύ trial & error κατέληξα να χρησιμοποιήσω το [rails_admin](#) gem ([helper video](#)) το οποίο στήνει τα πράγματα σχεδόν αυτόματα με λίγα απλά commands και υλοποιεί όλα όσα ζητάει η εκφώνηση και πολλά παραπάνω. Ο διαχειριστής έχει εγκατασταθεί στην εφαρμογή και τα credentials του δίνονται στην εισαγωγή της αναφοράς αυτής. Με το που κάνει login πηγαίνει κατευθείαν στο admin dashboard και δεν μπορεί κανένας άλλος χρήστης να έχει πρόσβαση στην σελίδα αυτή. (μπορείτε να προσπαθήσετε να δώσετε localhost:3000/admin και απλά θα κάνει redirect στο home page εκτός αν είστε συνδεδεμένοι με τον λογαριασμό του διαχειριστή)

Κάπου εδώ τελειώνει η αναφορά της εργασίας μου και παρακάτω στο κεφάλαιο 3 θα δώσω όλα τα git logs μου, τα οποία δείχνουν όλη την δουλειά που έχει γίνει στο project βηματικά από το initial commit έως το last commit καθώς και θα φανεί σε ποιά αρχεία έχει προστεθεί / αλλαχθεί / υλοποιηθεί οτιδήποτε έχουμε συζητήσει σε αυτήν εδώ την αναφορά.

Ευχαριστώ.

Κεφάλαιο 3ο

Git Logs

Initial Commit!

Create a simple Rails Project with Bootstrap

1. install bootstrap gem

Create basic authentication

1. install devise gem

Building navbar with partial view

1. copy navbar static from bootstrap demo
2. create partial shared/navbar
3. add devise paths to navbar


Authentication with full name

1. generate migration AddFullnameToUser fullname:string
2. add validations
3. add permissions for full name in devise in application controller

Update authentication views

1. add bootstrap to devise forms
2. add name instead of email to shared/navbar

Gravatar

- 
1. Update root path
 2. Add avatar_url helper with css class for different size

Notification

1. Add toastr-rails gem
2. Add js and css for toastr
3. Add toastr for flash messages
4. Add toastr for devise messages to views

Sending transactional email with Gmail

1. Add gem figaro and configure -> info from comments
2. Add smtp configuration to development.rb
3. Change devise configuration
4. Add confirmable to user.rb
5. Add migration for confirmable
6. Problems with gmail solution in [Stack Overflow](#)


Sending transactional email with Mailgun

1. Create mailgun account
2. Comment out gmail settings
3. Add credentials to config/application.yml
4. Change email address to valid one in devise.rb (config.mailer_sender = ['enter_valid_email_address_here'](#))

Create Facebook app

Create social authentication

1. Add gem omniauth and omniauth-facebook
2. Generate migration AddFieldsToUser provider:string uid:string image:string
3. Add config.omniauth :facebook, 'API_KEY', 'API_SECRET', scope: 'email', info_fields: 'email, name' to devise.rb

- 
4. Change 'API_KEY' and 'API_SECRET' to ENV['fb_app_id'] and ENV['fb_app_secret']
 5. Add method self.from_omniauth(auth) to User
 6. Add controller omniauth_callbacks_controller.rb and paste contents from [devise/omniauth](#)
 7. Add button to views/devise/sessions/new.html.erb and registrations/new
 8. Modify routes.rb
 9. Modify helper method to display avatar from fb as well
 10. Solution to uninitialized Users [Stack Overflow](#)

Styling views

1. Add styles and google fonts

Create user profile page

1. Generate migration - add phone_number and description to User
2. Add route for users :show
3. Create file users_controller.rb
4. Add styling for user show view

Create edit profile page

1. Add styles and additional fields to profile page
2. Remove current password_field from registrations/edit
3. Create RegistrationsController

Create Room Model

1. Rails g model Room with many fields
2. Add association to models

Create Room Controller

1. Rails g controller Rooms with many actions
2. Fill out Rooms controller actions

- 
3. Update routes.rb

Create Room Views

1. Create rooms/new view
2. Create sidebar
3. Create rooms/photo_upload view -> without uploads yet
4. Create rooms/listing view placeholder
5. Create rooms/amenities view placeholder
6. Create rooms/description view placeholder
7. Create rooms/location view placeholder
8. Create rooms/pricing view placeholder
9. Create partial for sidebar (link helpers are without @room and they are member routes)

Styling Views

1. Add styling to application.scss for form elements
2. Change class name for _room_menu.html.erb partial
3. Correct few style mistakes
4. Add some fronted validation to views

Install paperclip

1. Install ImageMagick "sudo apt-get install imagemagick"
2. Add gem paperclip

Create Photo Model

1. rails g model Photo room:references
2. rails g paperclip photo image
3. Add association to rooms and photos models
4. Add configuration to photo.rb from github gem page
5. Add routes for photos in rooms for create and destroy

Create Photo Controller

1. Create photos_controller.rb
2. Modify rooms_controller
3. Add before_action to check if user is room owner

Create Photo View

1. Add form fields for file uploads for photos
2. Add _photos_list.html.erb partial for showing all photos
3. Add some styling for photos upload form

Remove Photos with AJAX

1. Add rule in .gitignore to ignore photos
2. Add font awesome icons by adding link in head
3. Add link for removing photos in photos_list partial
4. Add destroy action to remove photos
5. Create photos/destroy.js.erb
6. Add http"S" to application helper for FB avatar

Amazon S3

1. Create new bucket on S3
2. Create new user IAM
3. Modify development.rb
4. Add keys to figaro
5. Add gem aws-sdk

Add Check to Room Views

1. Add check to sidebar in room creation and logic to display if form was submitted
2. Add publish button
3. Add logic only to display publish button when everything is field in and active is false

Update the Photo Removing with AJAX

1. Add ids to view
2. Add functionality to destroy.js.erb
3. Change room to class variable @room in photos_controller#destroy

Issue with hidden fields

1. In rooms_controller create private method
2. Merge hash with active: true if room is ready to prevent issue with changing code by user

Create Room Index Page

1. Create rooms/index.html.erb layout
2. Add default photo to listing if it doesn't have image
3. Copy blank.jpg to assets/images
4. Create method in model room.rb cover_photo

Create Room Show Page


1. Modify show.html.erb
2. Add toastr global config in application.js
3. Add styles for strikethrough

Add Google Map

1. Add gem geocoder
2. Add longitude and latitude to room
3. Add geocoded_by :address to room.rb model
4. Restart server and re enter address
5. Add script for google maps without API KEY

Add Nearby Rooms

1. Add nearby apartments to show.html.erb using geocoder

- 
2. Added code to enable map loading when changing to nearby room -> (errors in console to solve)
 3. Change units to km from miles in initializers/geocoder.rb

Create Reservations Model

1. rails g model Reservation user:references room:references start_date:datetime end_date:datetime price:integer total:integer
2. Add has_many :reservations to user and room
3. Add nested route for reservations

Create Reservations View

1. Add partial to room view
2. Add reservations/_form.html.erb

Create Reservations Controller

1. Add reservations_controller.rb

Add jQuery Date Picker


1. Add jquery-ui-rails gem
2. Add js require to application.js
3. Add script to _form.html.erb
4. Add stylesheet in tag for jquery ui theme

Refactoring Reservation Form

1. Add styles to right side booking form in room#show
2. Add styles for table in reservation form

AJAX for start date

1. Add preload method in rooms_controller to respond with json
2. Add route for rooms_controller#preload

- 
3. Add AJAX for retrieving reservations in `_form.html.erb` partial
 4. Add reservations to calendar
 5. Can't select date before and after reservation and remove disabled when selected

AJAX for end date

1. Add `is_conflict` private method in `rooms_controller`
2. Add action preview in `rooms_controller`
3. Add route preview
4. Add js logic in view `reservations/_form.html.erb` to handle request to `rooms/:id/preview`
5. Add js logic to display preview - nights, price, total etc
6. Add css for message

Create Your Trips Page

1. Add action `your_trips` in `reservations_controller`
2. Add route for trips
3. Create view for trips

Create Your Reservations Page

1. Add action `your_reservations` in `reservations_controller`
2. Add route
3. Add view for reservations
4. Copy sidebar from reservations to trips and listings
5. Update links in `shared/_navbar.html.erb`
6. Add button to navbar

Modify User Profile Page

1. Add `@rooms` to `users_controller`
2. Modify `show.html.erb` to show if user confirmed email or sign up with FB
3. Add rooms to profile

Creating Reviews Model

1. rails g model Review comment:text star:integer room:references reservation:references guest:references host:references type
2. rails g model GuestReview --parent=Review --migration=false
3. rails g model HostReview --parent=Review --migration=false
4. add default: 1 to star in migration
5. add belongs_to :guest, class_name: "User" to guest_review model
6. remove every belongs_to from review.rb model
7. Add has_many :guest_reviews to room.rb and helper method
8. Add has_many :guest_reviews, class_name: "GuestReview", foreign_key: "guest_id" to user.rb as well for Host

Creating Reviews Controller

1. Add host_reviews_controller.rb
2. Copy to guest_reviews_controller.rb
3. Add routes

Creating Reviews View

1. Get code for bootstrap modal
2. Create views/reviews/_guest_form.html.erb
3. Create views/reviews/_host_form.html.erb
4. Add button to your_trips with partial
5. Add button to your_reservations partial
6. Change modal id to be dynamic in views

Create Show Reviews Page

1. Add instance variable to rooms_controller
2. Add instance variable to users_controller to display reviews as a guest and as a host
3. Create reviews/_guest_list.html.erb
4. Create reviews/_host_list.html.erb

- 
5. Update room page and user page with comments partials

Adding jquery raty

1. Create jquery.raty.js and paste from [Raty](#)
2. Add images from folder from tutorial to assets/images

Add stars to reviews

1. Add stars to guest_form and host_form
2. Add stars to rooms/show.html.erb
3. Add stars to guest_list and host
4. Add id for modals

Update Home Page

1. Add @rooms to pages_controller.rb
2. Modify pages#home view
3. Add search form to home view
4. Add jquery for datepicker

Creating Search Page

1. In page controller add action search
2. Add search to routes.rb
3. Add file views/pages/search.html.erb
4. Add stylesheet rules for search page
5. Add search fields to view search.html.erb

Create search function

1. Add gem 'ransack'
2. Implement search in pages_controller.rb
3. Add code to search.html.erb to display results

Add Google Map

1. Transform search criteria to look better on search.html.erb
2. Add button to hide and open filters
3. Add google map with all found rooms
4. Add css for class of map markers
5. Add search form to navbar

AJAX Searching

1. Create rooms/_room_list.html.erb
2. Create /pages/search.js.erb
3. Change search.html.erb
4. Update google map with ajax

Add jquery pricing slider

1. Add require rule to assets/application.js
2. Add jquery to search.html.erb for sliders
3. Add some styling with jquery


Modify Home Page

1. Add images and h1 to home.html.erb
2. Add images to assets/images
3. Add stylesheet css rules

Improving Home Page

1. Change home page with partial
2. Change show user to use partial
3. Add some information to _rooms_list.html.erb partial
4. Add script for stars and reviews count

Auto Location Suggestion

- 
1. Use [geocomplete](#)
 2. Get google maps api key
 3. Add link to application.html.erb
 4. Create geocomplete.js file in javascripts and copy contents from geocomplete github
 5. Add id to form in home.html.erb and script
 6. Add css to hide scrollbars in search
 7. Add autolocation to manage listing

Dashboard Controller

1. Add dashboards_controller.rb
2. Add route for index action in dashboards_controller
3. Add rule for devise to redirect to dashboard after user sign in

Dashboard View

1. Add views/dashboards, add index.html.erb
2. Add link to user profile in rooms/show.html.erb
3. Add simple structure to the dashboard
4. Add style to navbar and more links in dropdown
5. Add additional navbar for sign in users on specific pages
6. Add active links to new navbar

Instant/Request Booking Model

1. rails g migration AddInstantToRooms instant:integer
2. rails g migration AddStatusToReservations status:integer
3. add code to migrations for default value
4. Add enum attributes to room and reservation model

Instant/Request Booking Function

1. Add :instant in room_params in rooms_controller
2. Add Booking Type select to rooms/new.html.erb
3. Add Booking Type select to rooms/listing.html.erb

- 4. Add logic to reservations_controller
- 5. Add different buttons to reservations/_form.html.erb

Approve/Decline Reservations

- 1. Create approve and decline in reservations_controller
- 2. Add routes for new actions
- 3. Add buttons for approving/declining in reservations/your_reservations
- 4. Show status in your_trips.html.erb

Reservation status

- 1. Add conditional to show review button only on approved reservations in views
- 2. Update logic on reservation form in rooms_controller
- 3. Update search action in pages_controller

Calendar Controller


- 1. Add gems fullcalendar-rails and moment js-rails [link](#)
- 2. Implement js and css in application.js and .scss
- 3. Create calendars_controller.rb
- 4. Add routes

Host Calendar Page

- 1. Create calendars/host.html.erb
- 2. Correct image display in calendar controller
- 3. Add css for calendar

Improving Host Calendar

- 1. Add date checking in js in host.html.erb for dates so the reservation will display correctly
- 2. Add search form for changing rooms
- 3. Add logic to controller for ransack search
- 4. Asynchronous reload of calendar. Add remote to form in view.

- 
5. Add host.je.erb

Calendar Next/Back

1. Add js to host.html.erb to trigger on change when pressing next/prev buttons
2. Add host calendar link to navbar

Calendar availability model

1. Add price display to host calendar
2. Add css for price
3. rails g model Calendar day:date price:integer status:integer room:references
4. Add relation to room.rb
5. Add enum to calendar.rb
6. Add routes as nested resource in rooms

Calendar Availability Form

1. Add /calendars/_form.html.erb
2. Add form to host calendar
3. Add js code for opening modal

Calendar Pricing

1. Add style for radio buttons
2. Add create action and calendar_params to calendars_controller
3. Modify host action in calendars_controller
4. Add in dayRender function in host.html.erb to add days data to calendar
5. Add reloading for arrows to host.js.erb

Sending Email for a Successful Booking (could not get it to work)

1. Create mailers/reservation_mailer.rb - getemoji.com
2. Create folder in views/reservation_mailer
3. Create file send_email_to_guest.html.erb

- 
4. Add line for sending email in `reservations_controller`

Conversations and Messages Model

1. rails g model Conversation sender_id:integer recipient_id:integer
2. rails g model Message context:text user:references conversation:references
3. Add relations, scopes and validations to `conversation.rb` model
4. Add validation to `message.rb` model and add conversion of time

Conversations and Messages Controller

1. Create file `conversations_controller.rb`
2. Create `messages_controller.rb`
3. Add routes for controllers

Conversations and Messages View


1. Create `conversations` folder and `index.html.erb`
2. Create `messages` folder and `index.html.erb`
3. Add link to conversations in `users/show` view
4. Add message link in `shared/navbar`
5. Some fixes of typos content changed to context
6. Add partial for messages
7. Change displaying of time to `time_ago_in_words`

Action Cable Configuration

1. In `routes.rb` mount ActionCable server
2. Establish socket connection for client side in `application.js`
3. Specify socket uri in `config/environments/development.rb`
4. Add ActionCable metatag in `application.html.erb`

Real time messages

1. rails g channel messages
2. add `stream_from` to `channels/messages_channel.rb`

- 
3. Define when should channel broadcast in messages_controller.rb
 4. Modify assets/javascripts/channels/messages.coffee
 5. In messages/index.html.erb add remote: true
 6. Change messages coffee to incorporate \$() to load javascript better

Notification Model

1. rails g model Notification content user:references
2. rails g migration AddUnreadToUser unread:integer
3. Add default: 0 to unread migration
4. Add relationships to user.rb model
5. Add after_create_commit to notification.rb model
6. Add create_notification in message.rb model
7. Add create_notification in reservation.rb model

Notification Controller


1. Add notifications_controller.rb
2. rails g channel notifications
3. Add code to notifications.coffee
4. Modify notifications_channel.rb
5. rails g job notification
6. Modify jobs/notification_job.rb
7. Add route for notifications in routes.rb

Notification View

1. Add notifications folder
2. Add index.html.erb for notifications
3. Create partial for notification
4. Add Notification count to dashboard/index.html.erb
5. Add hidden field with user id to layouts/application.html.erb

Improving Notification View

1. Add icon to shared/_navbar

- 
2. Add css for new class in application.scss
 3. Add code to show up number of notifications to notifications.coffee

Update Room Searching Function

1. Update pages_controller.rb

Unavailable Dates on Date Pickers

1. Add logic to get unavailable_dates in rooms_controller
2. Add javascript to reservations/_form.html.erb
3. Update is_conflict method in rooms_controller
4. Clean up code in _form.html.erb

Last Commit! Done!

Conclusion

Όπως λοιπόν γίνεται αντιληπτό μετά από τα παραπάνω, η εργασία ήταν αρκετά απαιτητική. Οι δυσκολίες ήταν ουκ ολίγες για να αναφερθούν όλες εδώ, αλλά αξιοσημείωτη είναι η σελίδα του διαχειριστή (admin page) η οποία με δυσκόλεψε αρκετά στο setup της, καθώς στην ruby έπαιρνα πάρα πολλά redirect, post & get errors κάτι το οποίο είχε να κάνει με το admin_controller και γενικά με τους ελέγχους των ήδη εγγεγραμμένων χρηστών αλλά τελικά ένα rails gem ονόματι rails_admin έκανε τα πράγματα πολύ πιο εύκολα τελικά στο στήσιμό τους όπως έχει αναφερθεί και παραπάνω. Βέβαια κατά την διάρκεια της αντιμετώπισής των δυσκολιών η λήψη γνώσης η οποία ελήφθη ήταν άκρως επιβραβευτική!

Το fullStack το οποίο χρησιμοποιήθηκε λοιπόν για την υλοποίηση της εργασίας αποτελείται από:

- HTML, CSS, Sass, Javascript & Bootstrap όπου ακολουθεί το MVC μοντέλο **για το frontend**
- Ruby **για το backend**
- Ruby on rails & rails templating **για την επικοινωνία frontend & backend**
- sqlite3 **για την βάση δεδομένων**
- Καθώς και AJAX **για το searching** & jQuery **για το DOM manipulation και πιο γρήγορα interactions**

Οι περισσότερες δυσκολίες αντιμετωπίστηκαν μέσω του Ruby on rails [Documentation](#), & φυσικά του [stackoverflow](#), αλλά και through trial & error!

Ευχαριστώ!

Ανδρέας Παππάς

A.M. : 1115201500201

