ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή Εργασία

Τίτλος Πτυχιακής Εργασίας	Λογισμικό διαχείρισης κρατήσεων τουριστικών επιχειρήσεων Hospitality web management system
Ονοματεπώνυμο Φοιτητή	Χρήστος Λέσκος
Πατρώνυμο	Θοδωράκης
Αριθμός Μητρώου	Π18093
Επιβλέπων	Σακκόπουλος Ευάγγελος, Αναπληρωτής Καθηγητής

Ημερομηνία Παράδοσης **Σεπτέμβριος 2024**

Copyright ©

Πίνακας περιεχομένων

Πίνακας Εικόνων

Περίληψη

Η παρούσα πτυχιακή εργασία αποτελεί την υλοποίηση του **Vaccoon**, ένα web λογισμικό διαχείρισης κρατήσεων για ιδιοκτήτες τουριστικών επιχειρήσεων. Πιο συγκεκριμένα, οι τουριστικές επιχειρήσεις που υποστηρίζονται είναι:

- Δωμάτια διαμονής
- Εστιατόριο
- Ενοικίαση αυτοκινήτου

Συχνά, ιδιοκτήτες έχουν επιχειρήσεις όπου συνδυάζουν τις παραπάνω κατηγορίες. Στόχος του **Vaccoon** είναι η καλύτερη οργάνωση των κρατήσεων μεταξύ των επιχειρήσεων, καθώς ο χρήστης μπορεί να αποθηκεύει τους πελάτες του, τις ιδιοκτησίες του και τις κρατήσεις, με αποτέλεσμα να έχει ταχύτερη και καλύτερη εικόνα της κατάστασης. Μερικές από τις λειτουργίες που παρέχονται είναι:

1. Προβολή κατάστασης κρατήσεων
2. Αναλυτική προβολή ημερολογιακής διαθεσιμότητας ιδιοκτησιών
3. Γρήγορο **check-in** μέσω σάρωσης QR κωδικού που αποστέλλεται στο email του πελάτη.
4. Στατιστικά μέτρα των επιχειρήσεων, για καλύτερη εικόνα της πορείας τους
5. Επιπρόσθετα στοιχεία πελάτη, όπως εντύπωση και σχόλια για καλύτερο προγραμματισμό σε μελλοντική κράτηση.
6. Ημερήσια προβολή εκκρεμών κρατήσεων για **check-in** και **check-out**

Το Vaccoon στοχεύει στην οργάνωση και την εύκολη πλοήγηση για τους ιδιοκτήτες τουριστικών επιχειρήσεων, βοηθώντας τους να διαχειριστούν κρατήσεις, ιδιοκτησίες και πελάτες. Κάθε σελίδα του συστήματος είναι προσβάσιμη από διάφορα σημεία, διευκολύνοντας την εμπειρία του χρήστη.

Οι σελίδες επιτρέπουν στοχευμένες μεταβάσεις, ακόμα και μέσω γενικών αναζητήσεων, κάνοντας τη διαδικασία πιο απλή για τους χρήστες που δεν θυμούνται λεπτομέρειες. Για παράδειγμα, αν ο ιδιοκτήτης αναζητήσει μια ευρύτερη περίοδο, το σύστημα θα εμφανίσει αναλυτικά τις κρατήσεις και τη διαθεσιμότητα για κάθε ιδιοκτησία κάνοντας εύκολο να εντοπίσει συγκεκριμένες πληροφορίες στο διάστημα αυτό. Με ένα κλικ, ο χρήστης μπορεί να δει τις λεπτομέρειες μιας κράτησης και από εκεί να μεταβεί στη σελίδα του πελάτη που έκανε την κράτηση. Αυτό το μοτίβο επαναλαμβάνεται συχνά, ώστε ο χρήστης να μην χρειάζεται να θυμάται συγκεκριμένες διαδρομές για να βρει τις πληροφορίες που χρειάζεται.

Για την υλοποίηση της εργασίας χρησιμοποιήθηκαν οι τεχνολογίες **Spring Boot** για την ανάπτυξη και τον σχεδιασμό του API που θα εκτελεί τις λειτουργίες στη πλευρά του εξυπηρετητή, **PostgreSQL** για την οργανωμένη αποθήκευση των απαραίτητων δεδομένων, **ReactJS** [7] για τον σχεδιασμό της web εφαρμογής στην πλευρά του χρήστη και **Bootstrap** [5][6] για μια καλύτερη και πιο γνώριμη εμπειρία του χρήστη.

This thesis presents the implementation of Vaccoon, a web-based booking management software for tourism business owners. Specifically, the supported tourism businesses include:

- Accommodation rooms
- Restaurants
- Car rentals

Often, owners have businesses that combine the above categories. The goal of Vaccoon is to better organize bookings/reservations across these businesses, allowing users to store their customers, properties, and bookings/reservations, resulting in a faster and clearer overview of their status. Some of the provided features include:

1. Booking status display
2. Detailed calendar availability view of properties
3. Quick check-in via QR code scanning sent to the customer's email
4. Business statistics for a better overview of their performance
5. Additional customer details, such as impressions and comments for better planning of future bookings/reservations
6. Daily view of pending bookings/reservations for check-in and check-out

Vaccoon aims to organize and simplify navigation for tourism business owners, helping them manage bookings/reservations, properties, and customers. Each page of the system is accessible from various points, enhancing the user experience. The pages allow targeted transitions, even through non-specific searches, making the process easier for users who may not remember details. For example, if the owner searches for a broader period, the system will display detailed bookings and availability for each property, making it easy to find specific information within that timeframe. With one click, the user can view the details of a booking and then navigate to the page of the customer who made the booking. This pattern is frequently repeated, so the user does not need to remember specific paths to find the information they need.

For the implementation of this project, the following technologies were used: **Spring Boot** for developing and designing the API that performs server-side functions, **PostgreSQL** for organized storage of necessary data, **ReactJS** for designing the web application on the user side, and **Bootstrap** for a better and more familiar user experience.

1. Καταγραφή και Ανάλυση Απαιτήσεων

1.1. Εισαγωγή

Οι βασικές λειτουργίες της εφαρμογής, αφορούν την διαχείρισης ιδιοκτησιών πελατών και κρατήσεων. Πιο συγκεκριμένα, αφορά την νέα προσθήκη, προβολή λίστας, προβολή λεπτομερειών, επεξεργασία/ενημέρωση και διαγραφή αυτών. Η λειτουργίες αυτές μπορούν να γίνουν από διαφορετικά σημεία της εφαρμογής και με διαφορετικά σενάρια. Παράδειγμα, η προσθήκη νέας κράτησης μπορεί να γίνει πηγαίνοντας από την μπάρα πλοήγησης στην σελίδα των κρατήσεων, αλλά μπορεί να επιτευχθεί επίσης στη σελίδα των δωματίων, επιλέγοντας ένα διαθέσιμο δωμάτιο μία συγκεκριμένη χρονική περίοδο. Παρακάτω ακολουθεί αναλυτική περιγραφή των περιπτώσεων αυτών.

1.2. Εγγραφή

Use Case: Εγγραφή.

Pre-conditions: Έχει γίνει μετάβαση στη σελίδα σύνδεσης.

Post-conditions: Δημιουργία νέου χρήστη.

Primary Path:

1. **Ο Χρήστης** κάνει κλικ στο Create new account hyperlink στη Login σελίδα.
2. **Το Σύστημα** τον μεταβιβάζει στη σελίδα “Register”.
3. **Ο Χρήστης** συμπληρώνει την φόρμα εγγραφής με τα προσωπικά του στοιχεία.
4. **Ο Χρήστης** κάνει κλικ στο κουμπί υποβολής.
5. **Ο Χρήστης δημιουργήθηκε επιτυχώς.**

Exception Path:

1. **Υπάρχει ήδη χρήστης με αυτό το email.**
2. **Ο Χρήστης** κάνει κλικ στο κουμπί υποβολής.
3. **Το Σύστημα** εμφανίζει μήνυμα πως υπάρχει ήδη εγγεγραμμένος χρήστης με αυτό το email.
4. **Το use case τελειώνει ανεπιτυχώς.**

1.1. Σύνδεση:

Use Case: Σύνδεση.

Pre-conditions: Υπάρχει εγγεγραμμένος χρήστης.

Post-conditions: Επιτυχής εξακρίβωση στοιχείων σύνδεσης.

Primary Path:

1. **Ο Χρήστης** συμπληρώνει τα στοιχεία του στην φόρμα login.
2. **Ο Χρήστης** κάνει κλικ στην υποβολή.
3. **Το Σύστημα** πλοηγεί τον **Χρήστη** στην αρχική σελίδα.
4. **Το use case τελειώνει επιτυχώς.**

Exception Path:

3a. Τα στοιχεία του Χρήστη δεν βρέθηκαν.

1. **Το Σύστημα** εμφανίζει μήνυμα πως τα στοιχεία του χρήστη δεν εξακριβώθηκαν.
2. **Το use case τελειώνει ανεπιτυχώς.**

1.2. Ιδιοκτησίες

1.2.1. Προσθήκη νέας ιδιοκτησίας:

Use Case: Προσθήκη νέας ιδιοκτησίας.

Pre-conditions: Επιτυχής σύνδεση.

Post-conditions: Προσθήκη νέας ιδιοκτησίας χρήστη.

Primary Path:

1. **Ο Χρήστης** κάνει κλικ στο κουμπί του menu.
2. **Το Σύστημα** εμφανίζει τις διαθέσιμες επιλογές στο menu.
3. **Ο Χρήστης** κάνει κλικ στην επιθυμητή κατηγορία ιδιοκτησίας.
4. **Το Σύστημα** πλοηγεί **τον Χρήστη** στη σελίδα εμφάνισης όλων των ιδιοκτησιών της κατηγορίας αυτής.
5. **Ο Χρήστης** κάνει κλικ στο κουμπί νέας προσθήκης.
6. **Το Σύστημα** πλοηγεί **τον Χρήστη** στη σελίδα προσθήκης νέας ιδιοκτησίας.
7. **Ο Χρήστης** συμπληρώνει την φόρμα με τα στοιχεία της ιδιοκτησίας.
8. **Ο Χρήστης** κάνει κλικ στην υποβολή.
9. **Το Σύστημα** εμφανίζεις επιτυχές μήνυμα.
10. **Το use case ολοκληρώνεται με επιτυχία.**

Alternate Path 1:

1a. **Ο Χρήστης** κάνει κλικ στο κουμπί"More" της επιθυμητής κατηγορίας ιδιοκτησίας στην αρχική σελίδα.

1. **Συνέχισε από το βήμα 4. του Primary Path**

Alternate Path 2:

1b. **Ο Χρήστης** κάνει κλικ στο κουμπί "Add New" της επιθυμητής κατηγορίας ιδιοκτησίας στην αρχική σελίδα.

1. **Συνέχισε από το βήμα 4. του Primary Path**

Exception Path:

7a. Το Σύστημα εμφανίζει μήνυμα πως υπάρχει ήδη ιδιοκτησία με αυτό το όνομα για τον συνδεδεμένο χρήστη.

1. **Ο Χρήστης** κάνει κλικ στην υποβολή.
2. **Το use case τελειώνει ανεπιτυχώς.**

1.2.2. Επεξεργασία Ιδιοκτησίας:

Use Case: Επεξεργασία ιδιοκτησίας.

Pre-conditions: Ο χρήστης βρίσκεται στη σελίδα εμφάνισης όλων των ιδιοκτησιών και υπάρχει τουλάχιστον μία ιδιοκτησία στην επιθυμητή κατηγορία.

Post-conditions: Ενημέρωση ιδιοκτησίας του χρήστη.

Primary Path:

1. **Ο Χρήστης** επιλέγει ένα στοιχείο από την λίστα.
2. **Ο Χρήστης** κάνει κλικ στο κουμπί εμφάνισης του στοιχείου.
3. **Το Σύστημα** πλοηγεί **τον Χρήστη** στην σελίδα εμφάνισης της ιδιοκτησίας.
4. **Ο Χρήστης** κάνει κλικ στο κουμπί επεξεργασίας.
5. **Το Σύστημα** εμφανίζει την φόρμα επεξεργασίας, στην οποία υπάρχουν ήδη συμπληρωμένα τα στοιχεία του επιλεγμένου στοιχείου.
6. **Ο Χρήστης** αλλάζει τα επιθυμητά πεδία.
7. **Ο Χρήστης** κάνει κλικ στο κουμπί υποβολής.
8. **Το Σύστημα** εμφανίζει επιτυχές μήνυμα.
9. **Το use case ολοκληρώνεται επιτυχώς.**

Alternate Path 1:

2a. Ο Χρήστης μεταβαίνει στη σελίδα εμφάνισης της ιδιοκτησίας μέσω της σελίδας εμφάνισης στοιχείων κράτησης.

1. **Ο Χρήστης** κάνει κλικ στο όνομα της ιδιοκτησίας της κράτησης.
2. **Συνέχισε από το βήμα 3 του Primary Path.**

Exception Path:

7a. Το Σύστημα εμφανίζει μήνυμα πως το όνομα που έχει συμπληρωθεί υπάρχει ήδη σε άλλη ιδιοκτησία που ανήκει στον χρήστη.

1. **Ο Χρήστης** κάνει κλικ στην υποβολή.
2. **Το use case τελειώνει ανεπιτυχώς.**

1.2.3. Διαγραφή ιδιοκτησίας:

Use Case: Διαγραφή ιδιοκτησίας.

Pre-conditions: Ο χρήστης βρίσκεται στη σελίδα εμφάνισης όλων των ιδιοκτησιών και υπάρχει τουλάχιστον μία ιδιοκτησία στην επιθυμητή κατηγορία.

Post-conditions: Διαγραφή ιδιοκτησίας του χρήστη.

Primary Path:

1. **Ο Χρήστης** επιλέγει ένα στοιχείο από την λίστα.
2. **Ο Χρήστης** κάνει κλικ στο κουμπί εμφάνισης του στοιχείου.
3. **Το Σύστημα** πλοηγεί **τον Χρήστη** στην σελίδα εμφάνισης της ιδιοκτησίας.
4. **Ο Χρήστης** κάνει κλικ στο κουμπί διαγραφής.
5. **Το Σύστημα** εμφανίζει μήνυμα επιβεβαίωσης της ενέργειας διαγραφής.
6. **Ο Χρήστης** κάνει κλικ στο κουμπί διαγραφής στο μήνυμα.
7. **Το Σύστημα** πλοηγεί τον χρήστη στην σελίδα εμφάνισης όλων των ιδιοκτησιών.
8. **Το use case ολοκληρώνεται με επιτυχία.**

Alternate Path 1:

2a. Ο Χρήστης μεταβαίνει στη σελίδα εμφάνισης της ιδιοκτησίας μέσω της σελίδας εμφάνισης στοιχείων κράτησης.

3. **Ο Χρήστης** κάνει κλικ στο όνομα της ιδιοκτησίας της κράτησης.
4. **Συνέχισε από το βήμα 3 του Primary Path.**

1.3. Πελάτες

1.3.1. Προσθήκη πελάτη:

Use Case: Προσθήκη πελάτη.

Pre-conditions: Επιτυχής σύνδεση.

Post-conditions: Προσθήκη νέου πελάτη του χρήστη.

Primary Path:

1. **Ο Χρήστης** κάνει κλικ στο κουμπί του menu.
2. **Το Σύστημα** εμφανίζει τις διαθέσιμες επιλογές στο menu.
3. **Ο Χρήστης** κάνει κλικ στην επιλογή του πελάτη.
4. **Το Σύστημα** πλοηγεί **τον Χρήστη** στη σελίδα εμφάνισης όλων των πελατών.
5. **Ο Χρήστης** κάνει κλικ στο κουμπί νέας προσθήκης.
6. **Το Σύστημα** πλοηγεί **τον Χρήστη** στη σελίδα προσθήκης νέου πελάτη.
7. **Ο Χρήστης** συμπληρώνει την φόρμα με τα στοιχεία του πελάτη.
8. **Ο Χρήστης** κάνει κλικ στην υποβολή.
9. **Το Σύστημα** εμφανίζεις επιτυχές μήνυμα.
10. **Το use case ολοκληρώνεται με επιτυχία.**

Alternate Path 1:

3a. **Ο Χρήστης** προσθέτει νέο χρήστη κατά την υποβολή νέας κράτησης

1. **Ο Χρήστης** κάνει κλικ στην επιλογή της κράτησης.
2. **Το Σύστημα** πλοηγεί τον χρήστη στη σελίδα εμφάνισης όλων των κρατήσεων.
3. **Ο Χρήστης** πατάει το κουμπί νέας κράτησης.
4. **Το Σύστημα** πλοηγεί τον χρήστη στη σελίδα νέας κράτησης.
5. **Ο Χρήστης** συμπληρώνει τα στοιχεία της κράτησης και στη φόρμα του χρήστη, συμπληρώνει τα στοιχεία του νέου χρήστη.
6. **Ο Χρήστης** κάνει κλικ στην υποβολή.
7. **Το Σύστημα** δεν βρίσκει τον χρήστη αποθηκευμένο και δημιουργεί έναν καινούργιο.
8. **Το use case ολοκληρώνεται με επιτυχία.**

Exception Path:

7a. Το Σύστημα εμφανίζει μήνυμα πως υπάρχει ήδη πελάτης με αυτό το email ή αριθμό για τον συνδεδεμένο χρήστη.

1. **Ο Χρήστης** κάνει κλικ στην υποβολή.
2. **Το Σύστημα** εμφανίζει μήνυμα σφάλματος.
3. **Το use case τελειώνει ανεπιτυχώς.**

1.3.2. Επεξεργασία Πελάτη:

Use Case: Επεξεργασία ιδιοκτησίας.

Pre-conditions: Ο χρήστης βρίσκεται στη σελίδα εμφάνισης όλων των πελατών και υπάρχει τουλάχιστον ένας αποθηκευμένος πελάτης.

Post-conditions: Ενημέρωση στοιχείων πελάτη.

Primary Path:

1. **Ο Χρήστης** επιλέγει έναν πελάτη από την λίστα.
2. **Ο Χρήστης** κάνει κλικ στο κουμπί εμφάνισης του πελάτη.
3. **Το Σύστημα** πλοηγεί **τον Χρήστη** στην σελίδα εμφάνισης του πελάτη.
4. **Ο Χρήστης** κάνει κλικ στο κουμπί επεξεργασίας.
5. **Το Σύστημα** εμφανίζει την φόρμα επεξεργασίας, στην οποία υπάρχουν ήδη συμπληρωμένα τα στοιχεία του επιλεγμένου πελάτη.
6. **Ο Χρήστης** αλλάζει τα επιθυμητά πεδία.
7. **Ο Χρήστης** κάνει κλικ στο κουμπί υποβολής.
8. **Το Σύστημα** εμφανίζει επιτυχές μήνυμα.
9. **Το use case ολοκληρώνεται επιτυχώς.**

Alternate Path 1:

2a. Ο Χρήστης μεταβαίνει στη σελίδα εμφάνισης του πελάτη μέσω της σελίδας εμφάνισης στοιχείων κράτησης.

5. **Ο Χρήστης** κάνει κλικ στο όνομα του πελάτη της κράτησης.
6. **Συνέχισε από το βήμα 3 του Primary Path.**

Exception Path:

7a. Το Σύστημα εμφανίζει μήνυμα πως το email ή ο αριθμός που έχει συμπληρωθεί υπάρχει ήδη σε άλλων πελάτη που ανήκει στον χρήστη.

1. **Ο Χρήστης** κάνει κλικ στην υποβολή.
2. **Το Σύστημα** εμφανίζει μήνυμα σφάλματος
3. **Το use case τελειώνει ανεπιτυχώς.**

1.3.3. Διαγραφή πελάτη:

Use Case: Διαγραφή πελάτη.

Pre-conditions: Ο χρήστης βρίσκεται στη σελίδα εμφάνισης όλων των πελατών και υπάρχει τουλάχιστον ένας αποθηκευμένος πελάτης.

Post-conditions: Διαγραφή πελάτη του χρήστη.

Primary Path:

1. **Ο Χρήστης** επιλέγει έναν πελάτη από την λίστα.
2. **Ο Χρήστης** κάνει κλικ στο κουμπί εμφάνισης του πελάτη.
3. **Το Σύστημα** πλοηγεί **τον Χρήστη** στην σελίδα εμφάνισης του πελάτη.
4. **Ο Χρήστης** κάνει κλικ στο κουμπί διαγραφής.
5. **Το Σύστημα** εμφανίζει μήνυμα επιβεβαίωσης της ενέργειας διαγραφής.
6. **Ο Χρήστης** κάνει κλικ στο κουμπί διαγραφής στο μήνυμα.
7. **Το Σύστημα** πλοηγεί τον χρήστη στην σελίδα εμφάνισης όλων των πελατών.
8. **Το use case ολοκληρώνεται με επιτυχία.**

Alternate Path 1:

2a. Ο Χρήστης μεταβαίνει στη σελίδα εμφάνισης του πελάτη μέσω της σελίδας εμφάνισης στοιχείων κράτησης.

1. **Ο Χρήστης** κάνει κλικ στο όνομα του πελάτη της κράτησης.
2. **Συνέχισε από το βήμα 3 του Primary Path.**

1.4. Κρατήσεις

1.4.1. Προσθήκη νέας κράτησης:

Use Case: Προσθήκη νέας κράτησης.

Pre-conditions: Επιτυχής σύνδεση.

Post-conditions: Αποθήκευση νέας κράτησης.

Primary Path:

1. **Ο Χρήστης** κάνει κλικ στο κουμπί του menu.
2. **Το Σύστημα** εμφανίζει τις διαθέσιμες επιλογές στο menu.
3. **Ο Χρήστης** κάνει κλικ στην επιλογή της κράτησης.
4. **Το Σύστημα** πλοηγεί **τον Χρήστη** στη σελίδα εμφάνισης όλων των κρατήσεων.
5. **Ο Χρήστης** κάνει κλικ στο κουμπί νέας προσθήκης.
6. **Το Σύστημα** πλοηγεί **τον Χρήστη** στη σελίδα προσθήκης νέας κράτησης.
7. **Ο Χρήστης** επιλέγει περίοδο κράτησης.
8. **Το Σύστημα** επιστρέφει τις διαθέσιμες ιδιοκτησίες του χρήστη για την δηλωμένη περίοδο.
9. **Ο Χρήστης** συμπληρώνει το πλήθος των πελατών της κράτησης.
10. **Ο Χρήστης** συμπληρώνει την φόρμα με τα στοιχεία του πελάτη της κράτησης.
11. **Ο Χρήστης** κάνει κλικ στην υποβολή.
12. **Το Σύστημα** εμφανίζεις επιτυχές μήνυμα.
13. **Το use case ολοκληρώνεται με επιτυχία.**

Alternate Path 1:

1a. **Ο Χρήστης** κάνει κλικ στο κουμπί νέα κράτηση από την αρχική οθόνη.

3. **Ο Χρήστης** κάνει κλικ στο κουμπί νέα κράτηση στην αρχική σελίδα.
4. **Συνέχισε από το βήμα 7 του Primary Path.**

Alternate Path 2:

1b. Ο Χρήστης πηγαίνει στη σελίδα νέας κράτησης από την σελίδα εμφάνισης λίστας ιδιοκτησιών.

1. **Ο Χρήστης** επιλέγει περίοδο ελέγχου διαθεσιμότητας ιδιοκτησιών.
2. **Το Σύστημα** ανανεώνει την λίστα με τα στοιχεία τα οποία δεν είναι δεσμευμένα από κάποια κράτηση την δηλωμένη περίοδο.
3. **Ο Χρήστης** επιλέγει ένα στοιχείο από την λίστα.
4. **Ο Χρήστης** κάνει κλικ στο κουμπί νέας κράτησης
5. **Το σύστημα** πλοηγεί **τον Χρήστη** στη σελίδα νέας κράτησης με συμπληρωμένα τα πεδία της περιόδου και της ιδιοκτησίας.
6. **Συνέχισε από το βήμα 9 του Primary Path.**

Alternate Path 3:

1c. Ο Χρήστης πηγαίνει στη σελίδα νέας κράτησης από την σελίδα εμφάνισης λίστας πελατών.

1. **Ο Χρήστης** επιλέγει έναν πελάτη από την λίστα.
2. **Ο Χρήστης** κάνει κλικ στο κουμπί νέας κράτησης
3. **Το σύστημα** πλοηγεί **τον Χρήστη** στη σελίδα νέας κράτησης με συμπληρωμένα τα πεδία των στοιχείων του πελάτη.
4. **Συνέχισε από το βήμα 7 του Primary Path.**

Exception Path:

11a. Το Σύστημα εμφανίζει μήνυμα πως υπάρχει ήδη πελάτης με αυτό το email ή αριθμό για τον συνδεδεμένο χρήστη αλλά το email και ο αριθμός του πελάτη που βρέθηκε δεν συμβαδίζουν με αυτά που έδωσε ο χρήστης.

1. **Ο Χρήστης** κάνει κλικ στην υποβολή.
2. **Το Σύστημα** εμφανίζει μήνυμα σφάλματος.
3. **Το use case τελειώνει ανεπιτυχώς.**

1.4.2. Check in κράτησης:

Use Case: Check in κράτησης.

Pre-conditions: Υπάρχει τουλάχιστον μια αποθηκευμένη κράτηση με κατάσταση ‘PENDING’.

Post-conditions: Αλλαγή κατάστασης κράτησης σε ‘CHECKED IN’.

Primary Path:

1. **Ο Χρήστης** κάνει κλικ στο κουμπί του menu.
2. **Το Σύστημα** εμφανίζει τις διαθέσιμες επιλογές στο menu.
3. **Ο Χρήστης** κάνει κλικ στην επιλογή της κράτησης.
4. **Το Σύστημα** πλοηγεί **τον Χρήστη** στη σελίδα εμφάνισης όλων των κρατήσεων.
5. **Ο Χρήστης** επιλέγει μία κράτηση από την λίστα.
6. **Το Σύστημα** πλοηγεί **τον Χρήστη** στη σελίδα εμφάνισης των στοιχείων της κράτησης
7. **Ο Χρήστης** κάνει κλικ στο κουμπί Check in.
8. **Το Σύστημα** ανανεώνει τη κατάσταση της κράτησης σε 'CHECKED IN'
9. **Το use case ολοκληρώνεται με επιτυχία**

Alternate Path 1:

1a. Ο Χρήστης πηγαίνει στα στοιχεία της κράτησης σαρώνοντας το QR code της κράτησης.

1. **Ο Χρήστης** κάνει κλικ στο κουμπί σάρωσης.
2. **Το Σύστημα** πλοηγεί τον χρήστη στη σελίδα σάρωσης QR code.
3. **Ο Χρήστης** σαρώνει μπροστά στη κάμερα της συσκευής το QR code της κράτησης.
4. **Συνέχισε από το βήμα 6 του Primary Path.**

Alternate Path 2:

1b. Ο Χρήστης πηγαίνει στη σελίδα εμφάνισης στοιχείων κράτησης από την αρχική σελίδα.

1. **Το Σύστημα** εμφανίζει στην αρχική σελίδα τις κρατήσεις που ξεκινάνε από την σημερινή μέρα και δεν έχουν γίνει ακόμα Check in.
2. **Ο Χρήστης** κάνει κλικ σε μία από τις κρατήσεις που εμφανίζονται.
3. **Συνέχισε από το βήμα 6 του Primary Path.**

1.4.3. Check out κράτησης:

Use Case: Check out κράτησης.

Pre-conditions: Υπάρχει τουλάχιστον μια αποθηκευμένη κράτηση με κατάσταση 'CHECKED IN'.

Post-conditions: Αλλαγή κατάστασης κράτησης σε 'CHECKED OUT'.

Primary Path:

1. **Ο Χρήστης** κάνει κλικ στο κουμπί του menu.
2. **Το Σύστημα** εμφανίζει τις διαθέσιμες επιλογές στο menu.
3. **Ο Χρήστης** κάνει κλικ στην επιλογή της κράτησης.
4. **Το Σύστημα** πλοηγεί **τον Χρήστη** στη σελίδα εμφάνισης όλων των κρατήσεων.
5. **Ο Χρήστης** επιλέγει μία κράτηση από την λίστα.
6. **Το Σύστημα** πλοηγεί **τον Χρήστη** στη σελίδα εμφάνισης των στοιχείων της κράτησης
7. **Ο Χρήστης** κάνει κλικ στο κουμπί Check out.
8. **Το Σύστημα** ανανεώνει τη κατάσταση της κράτησης σε 'CHECKED OUT'.
9. **Το use case ολοκληρώνεται με επιτυχία**

Alternate Path 1:

1a. Ο Χρήστης πηγαίνει στα στοιχεία της κράτησης σαρώνοντας το QR code της κράτησης.

1. **Ο Χρήστης** κάνει κλικ στο κουμπί σάρωσης.
2. **Το Σύστημα** πλοηγεί τον χρήστη στη σελίδα σάρωσης QR code.
3. **Ο Χρήστης** σαρώνει μπροστά στη κάμερα της συσκευής το QR code της κράτησης.
4. **Συνέχισε από το βήμα 6 του Primary Path.**

Alternate Path 2:

1b. Ο Χρήστης πηγαίνει στη σελίδα εμφάνισης στοιχείων κράτησης από την αρχική σελίδα.

1. **Το Σύστημα** εμφανίζει στην αρχική σελίδα τις κρατήσεις που λήγουν την σημερινή μέρα και δεν έχουν γίνει ακόμα Check out.
2. **Ο Χρήστης** κάνει κλικ σε μία από τις κρατήσεις που εμφανίζονται.
3. **Συνέχισε από το βήμα 6 του Primary Path.**

1.4.4. Ακύρωση κράτησης:

Use Case: Ακύρωση κράτησης.

Pre-conditions: Υπάρχει τουλάχιστον μια αποθηκευμένη κράτηση με κατάσταση 'PENDING'.

Post-conditions: Αλλαγή κατάστασης κράτησης σε 'CANCELED'.

Primary Path:

1. **Ο Χρήστης** κάνει κλικ στο κουμπί του menu.
2. **Το Σύστημα** εμφανίζει τις διαθέσιμες επιλογές στο menu.
3. **Ο Χρήστης** κάνει κλικ στην επιλογή της κράτησης.
4. **Το Σύστημα** πλοηγεί **τον Χρήστη** στη σελίδα εμφάνισης όλων των κρατήσεων.
5. **Ο Χρήστης** επιλέγει μία κράτηση από την λίστα.
6. **Το Σύστημα** πλοηγεί **τον Χρήστη** στη σελίδα εμφάνισης των στοιχείων της κράτησης
7. **Ο Χρήστης** κάνει κλικ στο κουμπί Cancel.
8. **Το Σύστημα** ανανεώνει τη κατάσταση της κράτησης σε 'CANCELED'
9. **Το use case ολοκληρώνεται με επιτυχία**

Alternate Path 1:

1a. Ο Χρήστης πηγαίνει στα στοιχεία της κράτησης σαρώνοντας το QR code της κράτησης.

1. **Ο Χρήστης** κάνει κλικ στο κουμπί σάρωσης.
2. **Το Σύστημα** πλοηγεί τον χρήστη στη σελίδα σάρωσης QR code.
3. **Ο Χρήστης** σαρώνει μπροστά στη κάμερα της συσκευής το QR code της κράτησης.
4. **Συνέχισε από το βήμα 6 του Primary Path.**

Alternate Path 2:

1b. Ο Χρήστης πηγαίνει στη σελίδα εμφάνισης στοιχείων κράτησης από την αρχική σελίδα.

1. **Το Σύστημα** εμφανίζει στην αρχική σελίδα τις κρατήσεις που ξεκινάνε από την σημερινή μέρα και δεν έχουν γίνει ακόμα Check in.
2. **Ο Χρήστης** κάνει κλικ σε μία από τις κρατήσεις που εμφανίζονται.
3. **Συνέχισε από το βήμα 6 του Primary Path.**

1.4.5. Επεξεργασία Κράτησης:

Use Case: Επεξεργασία κράτησης.

Pre-conditions: Υπάρχει τουλάχιστον μια αποθηκευμένη κράτηση.

Post-conditions: Ενημέρωση στοιχείων κράτησης.

Primary Path:

1. **Ο Χρήστης** επιλέγει μια κράτηση από την λίστα.
2. **Ο Χρήστης** κάνει κλικ στο κουμπί εμφάνισης της κράτησης.
3. **Το Σύστημα** πλοηγεί **τον Χρήστη** στην σελίδα εμφάνισης της κράτησης.
4. **Ο Χρήστης** κάνει κλικ στο κουμπί επεξεργασίας.
5. **Το Σύστημα** εμφανίζει την φόρμα επεξεργασίας, στην οποία υπάρχουν ήδη συμπληρωμένα τα στοιχεία της επιλεγμένης κράτησης.
6. **Ο Χρήστης** αλλάζει τα επιθυμητά πεδία.
7. **Ο Χρήστης** κάνει κλικ στο κουμπί υποβολής.
8. **Το Σύστημα** εμφανίζει επιτυχές μήνυμα.
9. **Το use case ολοκληρώνεται επιτυχώς.**

Alternate Path 1:

1a. Ο Χρήστης πηγαίνει στα στοιχεία της κράτησης σαρώνοντας το QR code της κράτησης.

1. **Ο Χρήστης** κάνει κλικ στο κουμπί σάρωσης.
2. **Το Σύστημα** πλοηγεί τον χρήστη στη σελίδα σάρωσης QR code.
3. **Ο Χρήστης** σαρώνει μπροστά στη κάμερα της συσκευής το QR code της κράτησης.
4. **Συνέχισε από το βήμα 3 του Primary Path.**

Alternate Path 2:

1b. Ο Χρήστης πηγαίνει στη σελίδα εμφάνισης στοιχείων κράτησης από την αρχική σελίδα.

1. **Το Σύστημα** εμφανίζει στην αρχική σελίδα τις κρατήσεις που ξεκινάνε και που λήγουν από την σημερινή μέρα και δεν έχουν γίνει ακόμα Check in ή Check out αντίστοιχα.
2. **Ο Χρήστης** κάνει κλικ σε μία από τις κρατήσεις που εμφανίζονται.

3. **Συνέχισε από το βήμα 3 του Primary Path.**

1.4.6. Διαγραφή κράτησης:

Use Case: Διαγραφή κράτησης.

Pre-conditions: Υπάρχει τουλάχιστον μια αποθηκευμένη κράτηση.

Post-conditions: Διαγραφή κράτησης του χρήστη.

Primary Path:

1. **Ο Χρήστης** επιλέγει μια κράτηση από την λίστα.
2. **Ο Χρήστης** κάνει κλικ στο κουμπί εμφάνισης της κράτησης.
3. **Το Σύστημα** πλοηγεί **τον Χρήστη** στην σελίδα εμφάνισης της κράτησης.
4. **Ο Χρήστης** κάνει κλικ στο κουμπί διαγραφής της κράτησης.
5. **Το Σύστημα** εμφανίζει μήνυμα επιβεβαίωσης της ενέργειας.
6. **Ο Χρήστης** κάνει κλικ στο κουμπί διαγραφή, του μηνύματος.
7. **Το use case ολοκληρώνεται επιτυχώς.**

Alternate Path 1:

1a. Ο Χρήστης πηγαίνει στα στοιχεία της κράτησης σαρώνοντας το QR code της κράτησης.

1. **Ο Χρήστης** κάνει κλικ στο κουμπί σάρωσης.
2. **Το Σύστημα** πλοηγεί τον χρήστη στη σελίδα σάρωσης QR code.
3. **Ο Χρήστης** σαρώνει μπροστά στη κάμερα της συσκευής το QR code της κράτησης.
4. **Συνέχισε από το βήμα 3 του Primary Path.**

Alternate Path 2:

1b. Ο Χρήστης πηγαίνει στη σελίδα εμφάνισης στοιχείων κράτησης από την αρχική σελίδα.

1. **Το Σύστημα** εμφανίζει στην αρχική σελίδα τις κρατήσεις που ξεκινάνε και που λήγουν από την σημερινή μέρα και δεν έχουν γίνει ακόμα Check in ή Check out αντίστοιχα.
2. **Ο Χρήστης** κάνει κλικ σε μία από τις κρατήσεις που εμφανίζονται.
3. **Συνέχισε από το βήμα 3 του Primary Path.**

1.4.7. Εμφάνιση στατιστικών κρατήσεων:

Use Case: Εμφάνιση στατιστικών κρατήσεων.

Pre-conditions: Υπάρχει τουλάχιστον μια αποθηκευμένη κράτηση.

Post-conditions: Πλοήγηση στη σελίδα στατιστικών στοιχείων των κρατήσεων του χρήστη.

Primary Path:

1. **Ο Χρήστης** κάνει κλικ στο κουμπί του menu.
2. **Το Σύστημα** εμφανίζει τις διαθέσιμες επιλογές στο menu.
3. **Ο Χρήστης** κάνει κλικ στην επιλογή της κράτησης.
4. **Το Σύστημα** πλοηγεί **τον Χρήστη** στη σελίδα εμφάνισης όλων των κρατήσεων.
5. **Ο Χρήστης** κάνει κλικ στο κουμπί εμφάνισης στατιστικών.
6. **Το Σύστημα** πλοηγεί **τον Χρήστη** στη σελίδα εμφάνισης των στατιστικών των κρατήσεων.
7. **Το use case ολοκληρώνεται με επιτυχία.**

2. Τεχνική ανάλυση υλοποίησης API

2.1. Εισαγωγή

2.1.1. Μοντέλο User

Το User model, υλοποιεί το **UserDetails** interface του Spring security και περιέχει τα απαραίτητα στοιχεία σύνδεσης και λειτουργίας του χρήστη.

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "user_table")
public class User implements UserDetails {

    @Id
    @GeneratedValue
    private Integer id;
    private String firstname;
    private String lastname;

    @Column(unique = true)
    private String email;
    private String password;

    @OneToMany(mappedBy = "user")
    private List<Token> tokens;

    @OneToMany(mappedBy = "user")
    private List<Property> properties;
```

1. Μοντέλο User

Πιο συγκεκριμένα, βρίσκουμε τα πεδία **id**, **firstname**, **lastname**, **email**, **password**, **tokens** και **properties**. Βλέπουμε πως τα δύο τελευταία έχουν γίνει annotated ως OneToMany. Αυτό γίνεται επειδή το id του user χρησιμοποιείται ως foreign key στις κλάσεις αυτές. Σημαντικό να σημειωθεί πως για τους getters και setters των κλάσεων, χρησιμοποιούμε την βιβλιοθήκη του **Lombok** η οποία τα παρέχει χωρίς να χρειαστεί να τα γράψουμε οι ίδιοι, με το annotation Data πάνω από την κλάση.

2.1.2. User Repository

Το UserRepository, όπως και τα υπόλοιπα Repositories που θα δούμε στην εφαρμογή, είναι ένα interface που επεκτείνει το **JpaRepository**, το οποίο παρέχει κάποιες μεθόδους για την διαχείριση των δεδομένων στην βάση δεδομένων μας καθώς και τα query methods, όπου είναι μέθοδοι που παράγουν το query που θα εκτελεστεί αυτόματα, εάν ακολουθούμε μία σύμβαση ονομασίας όταν τις δημιουργούμε.

```
@Repository
public interface UserRepository extends JpaRepository<User, Integer> {

    4 usages  chrisleskos
    Optional<User> findByEmail(String email);

}
```

2. User Repository

Στην φωτογραφία μπορούμε να δούμε πως έχουμε βάλει το Repository annotation πάνω από την κλάση, όπως ορίζεται στο Spring Framework. Όπως προαναφέρθηκε, το JpaRepository interface, μας παρέχει ήδη μεθόδους αποθήκευσης και συνολικής συλλογής δεδομένων προς και από την βάση αντίστοιχης. Για την συλλογή χρηστών με δεδομένο ένα συγκεκριμένο email, έχει δημιουργηθεί η **findByEmail** query method, η οποία από το όνομα που της έχει δοθεί, γνωρίζει πως πρέπει να κατασκευάσει το query που θα εκτελεστεί.

Η δημιουργία του API έγινε με σκοπό να δημιουργηθεί μία υπηρεσία υλοποίησης της λογικής της εφαρμογής διαχωρίζοντάς την τελείως από το γραφικό περιβάλλον και την προβολή των αποτελεσμάτων. Έτσι, στο Vaccoon API συμπεριλαμβάνονται οι λειτουργίες δημιουργίας και διαχείρισης συνδέσεων και πιστοποίησης των χρηστών καθώς και η διαχείριση ιδιοκτησιών, πελατών, κρατήσεων και στατιστικών του χρήστη, επιστρέφοντας την απαραίτητη πληροφορία σε μορφή **JSON**.

Κατά την αυθεντικοποίηση των στοιχείων, επιστρέφεται ένα **JSON Web Token** (JWT) το οποίο είναι ενεργό για συγκεκριμένο διάστημα και είναι απαραίτητο να συμπεριλαμβάνεται σε οποιαδήποτε κλήση κάνουμε στο API με εξαίρεση ορισμένων. Το JWT πρέπει να συμπεριληφθεί στο header του αιτήματος δηλώνοντας το Authorization ως Bearer:

```
Authorization: Bearer <token>
```

3. Παράδειγμα σύνταξης του Authorization Bearer

Σημαντική σημείωση είναι επίσης, πως έχει χρησιμοποιηθεί η **MVC** αρχιτεκτονική και η οργάνωση των αρχείων δεν έχει γίνει ανά τύπο αρχείου αλλά ανά *model*. Αυτό σημαίνει ενδεικτικά, πως μέσα στον φάκελο **client** είναι πιθανό να βρούμε τα εξής αρχεία:

- Client
- ClientController
- ClientService
- ClientRepository

Τέλος, η ασφάλεια της εφαρμογής έχει υλοποιηθεί με το **Spring Security.**

2.2. User

2.2.1. Μοντέλο User

Το User model, υλοποιεί το **UserDetails** interface του Spring security και περιέχει τα απαραίτητα στοιχεία σύνδεσης και λειτουργίας του χρήστη.

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "user_table")
public class User implements UserDetails {

    @Id
    @GeneratedValue
    private Integer id;
    private String firstname;
    private String lastname;

    @Column(unique = true)
    private String email;
    private String password;

    @OneToMany(mappedBy = "user")
    private List<Token> tokens;

    @OneToMany(mappedBy = "user")
    private List<Property> properties;
```

4. Μοντέλο User

Πιο συγκεκριμένα, βρίσκουμε τα πεδία **id**, **firstname**, **lastname**, **email**, **password**, **tokens** και **properties**. Βλέπουμε πως τα δύο τελευταία έχουν γίνει annotated ως OneToMany. Αυτό γίνεται επειδή το id του user χρησιμοποιείται ως foreign key στις κλάσεις αυτές. Σημαντικό να σημειωθεί πως για τους getters και setters των κλάσεων, χρησιμοποιούμε την βιβλιοθήκη του **Lombok** η οποία τα παρέχει χωρίς να χρειαστεί να τα γράψουμε οι ίδιοι, με το annotation Data πάνω από την κλάση.

2.2.2. User Repository

Το UserRepository, όπως και τα υπόλοιπα Repositories που θα δούμε στην εφαρμογή, είναι ένα interface που επεκτείνει το **JpaRepository**, το οποίο παρέχει κάποιες μεθόδους για την διαχείριση των δεδομένων στην βάση δεδομένων μας καθώς και τα query methods, όπου είναι μέθοδοι που παράγουν το query που θα εκτελεστεί αυτόματα, εάν ακολουθούμε μία σύμβαση ονομασίας όταν τις δημιουργούμε.

```
@Repository
public interface UserRepository extends JpaRepository<User, Integer> {

    4 usages  chrisleskos
    Optional<User> findByEmail(String email);

}
```

5. User Repository

Στην φωτογραφία μπορούμε να δούμε πως έχουμε βάλει το Repository annotation πάνω από την κλάση, όπως ορίζεται στο Spring Framework. Όπως προαναφέρθηκε, το JpaRepository interface, μας παρέχει ήδη μεθόδους αποθήκευσης και συνολικής συλλογής δεδομένων προς και από την βάση αντίστοιχης. Για την συλλογή χρηστών με δεδομένο ένα συγκεκριμένο email, έχει δημιουργηθεί η **findByEmail** query method, η οποία από το όνομα που της έχει δοθεί, γνωρίζει πως πρέπει να κατασκευάσει το query που θα εκτελεστεί.

2.3. Token

2.3.1. Μοντέλο Token

Το Token είναι το model που αντιπροσωπεύει την διαχείριση των JWT της εφαρμογής.

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
public class Token {

  @Id
  @GeneratedValue
  public Integer id;

  @Column(unique = true)
  public String token;

  @Enumerated(EnumType.STRING)
  public TokenType tokenType = TokenType.BEARER;

  public boolean revoked;

  public boolean expired;

  @ManyToOne(fetch = FetchType.LAZY)
  @JoinColumn(name = "user_id")
  public User user;
}
```

6. Μοντέλο Token

Τα fields του είναι το **id** ως αναγνωριστικό, το **token** όπου είναι η τιμή του JWT, το **tokenType** είναι ως προεπιλογή Bearer, την τιμή του οποίου παίρνουμε από ένα Enum που έχουμε φτιάξει για αυτόν τον σκοπό, τα **revoked** και **expired** τα οποία

χρησιμεύουν για τον έλεγχο εγκυρότητας του token, σε συνδυασμό πάντα με τον **user** στον οποίο ανήκει.

2.3.2. Token Repository

Όμοια και εδώ, επεκτείνουμε το JpaRepository.

```
@Repository
public interface TokenRepository extends JpaRepository<Token, Integer> {

  1 usage  chrisleskos
  @Query(value = """
      select t from Token t inner join User u\s
      on t.user.id = u.id\s
      where u.id = :id and (t.expired = false or t.revoked = false)\s
      """)
  List<Token> findAllValidTokenByUser(Integer id);

  3 usages  chrisleskos
  Optional<Token> findByToken(String token);
}
```

7. Token Repository

Εδώ έχει χρησιμοποιηθεί το Query annotation στην μέθοδο **findAllValidTokenByUser**, για να δηλώσουμε πως το Query που θα χρησιμοποιήσουμε, καθώς είναι πιο περίπλοκο και ένα query method [3] δεν θα βοηθούσε καθώς η συμβατική ονομασία δεν επιτρέπει αρκετή λεπτομέρεια. Το value του annotation λέγεται **JPQL** και είναι μία ιδιαίτερη περίπτωση query που μοιάζει αρκετά στη συνηθισμένη SQL που χρησιμοποιούμε στα περισσότερα γνωστά Συστήμα σχεσιακών βάσεων δεδομένων. Η σημαντική διαφορά είναι, πως στην JPQL αναφερόμαστε, στις εγγραφές και τα columns του πίνακα, μέσω των μοντέλων και των fields, αντίστοιχα, που τα αντιπροσωπεύουν στην εφαρμογή.

2.4. Authentication

2.4.1. Registration Request

Η **RegisterRequest** κλάση συγκεντρώνει τα δεδομένα που καταχωρεί ο χρήστης όταν εκτελεί την διαδικασία εγγραφής του στο σύστημα.

```
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class RegisterRequest {
    private String firstname;
    private String lastname;
    private String email;
    private String password;
}
```

8. Register Request

Τα fields συγκεκριμένα είναι **firstname**, **lastname**, **email** και **password**.

2.4.2. Authentication Request

Παρομοίως με την RegisterRequest, η **AuthenticationRequest** συγκεντρώνει τα δεδομένα που καταχωρεί ο χρήστης στην διαδικασία της σύνδεσης στο σύστημα.

```java
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class AuthenticationRequest {
    private String email;
    private String password;
}
```

9. Authentication Request

Τα δεδομένα αυτά είναι το **email** και το **password**.

2.4.3. Authentication Response

Αφού η αυθεντικοποίηση ολοκληρωθεί επιτυχώς, το σύστημα απαντάει με ένα αντικείμενο της **AuthenticationResponse** κλάσης. Σε αυτή θα βρούμε το **token** που παράχθηκε, το **firstname**, **lastname** και το **username** (email) του χρήστη.

```java
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class AuthenticationResponse {
    private String token;
    private String refreshToken;
    private String firstname;
    private String lastname;
    private String username;
}
```

10. Authentication Response

2.4.4. Authentication Service

Στην AuthenticationService κλάση, βρίσκεται η λογική πίσω από όλες τις ενέργειες που αφορούν την πιστοποίηση και την πρόσβαση του χρήστη στην εφαρμογή.

```
@Service
@RequiredArgsConstructor
public class AuthenticationService {

    1 usage
    Logger log = Logger.getLogger(AuthenticationService.class.getName());
    private final UserDetailsService userDetailsService;
    private final UserRepository userRepository;
    private final TokenRepository tokenRepository;
    private final PasswordEncoder passwordEncoder;
    private final JwtService jwtService;
    private final AuthenticationManager authenticationManager;
```

11. Πεδία του Authentication Service

Για την υλοποίηση χρησιμοποιεί την **UserDetailsService** κλάση από το Spring Security που αφορά τα στοιχεία του χρήστη της εφαρμογής, το **UserRepository** και **TokenRepository** που αναλύθηκαν πιο πάνω, το **PasswordEncoder** και συγκεκριμένα το Bcrypt για την κρυπτογράφηση των κωδικών που θα αποθηκευτούν και αργότερα θα πιστοποιηθούν, **JwtService** μία service κλάση της εφαρμογής η οποία εκτελεί την λογική δημιουργίας, αποθήκευσης, διαγραφής και ελέγχου εγκυρότητας των JWT και τέλος το **AuthenticationManager** , επίσης μία κλάση του Spring Security για τον έλεγχο πιστοποίησης των χρηστών.

Η **register** μέθοδος, θα δημιουργήσει στην βάση έναν καινούργιο χρήστη, ένα καινούργιο JWT και θα επιστρέψει ένα AuthenticationResponse αντικείμενο.

Πιο συγκεκριμένα, από το RegisterRequest αντικείμενο που θα δοθεί, φορτώνουμε σε ένα αντικείμενο χρήστη το **firstname**, **lastname** και **email** του και κάνουμε encode χρησιμοποιώντας το PasswordEncoder, τον **κωδικό**.

```
public AuthenticationResponse register(RegisterRequest request) {
    var user = User.builder()
            .firstname(request.getFirstname())
            .lastname(request.getLastname())
            .email(request.getEmail())
            .password(passwordEncoder.encode(request.getPassword()))
            .build();

    var savedUser = userRepository.save(user);
    var jwt = jwtService.generateToken(user);
    saveUserToken(savedUser, jwt);

    return AuthenticationResponse.builder()
            .token(jwt)
            .firstname(user.getFirstname())
            .lastname(user.getLastname())
            .username(user.getUsername())
            .build();
}
```

12. register μέθοδος του Authentication Service

Αποθηκεύουμε τον χρήστη αυτόν και τον χρησιμοποιούμε για να δημιουργήσουμε ένα token το οποίο και αποθηκεύουμε.

```
private void saveUserToken(User user, String jwt) {
    var token = Token.builder()
            .user(user)
            .token(jwt)
            .tokenType(TokenType.BEARER)
            .expired(false)
            .revoked(false)
            .build();
    tokenRepository.save(token);
}
```

13. saveUserToken μέθοδος του Authentication Service

Η **authenticate** μέθοδος, είναι υπεύθυνη για τον έλεγχο του χρήστη κατά την σύνδεση. Πρώτα χρησιμοποιούμε το AuthenticationManager και την authenticate μέθοδο για να ελέγξουμε ότι το email και ο κωδικός είναι έγκυρα. Εάν δεν είναι, το Spring Security πετάει Authentication Exception.

```
public AuthenticationResponse authenticate(AuthenticationRequest request) {
    authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(request.getEmail(), request.getPassword())
    );

    var user = userRepository.findByEmail(request.getEmail())
            .orElseThrow();

    var jwt = jwtService.generateToken(user);
    revokeAllUserTokens(user);
    saveUserToken(user, jwt);

    return AuthenticationResponse.builder()
            .token(jwt)
            .firstname(user.getFirstname())
            .lastname(user.getLastname())
            .username(user.getUsername())
            .build();
}
```

14. authenticate μέθοδος του Authentication Service

Εάν είναι έγκυρα, φορτώνουμε τον χρήστη με αυτό το email σε ένα User αντικείμενο, δημιουργούμε και αποθηκεύουμε ένα token με τα στοιχεία του και θέτουμε όλα τα υπάρχοντα tokens του χρήστη ως μη έγκυρα.

```
private void revokeAllUserTokens(User user) {
    var validUserTokens = tokenRepository.findAllValidTokenByUser(user.getId());
    if (validUserTokens.isEmpty())
        return;
    validUserTokens.forEach(token -> {
        token.setExpired(true);
        token.setRevoked(true);
    });
    tokenRepository.saveAll(validUserTokens);
}
```

15. revokeAllUserTokens μέθοδος του Authentication Service

Στην μέθοδο **checkIfLogged**, ελέγχουμε εάν το JWT που παρέχεται στις κλήσεις της εφαρμογής, είναι έγκυρο και ανήκει σε έναν συνδεδεμένο χρήστη. Περνάμε σαν παράμετρο το request που δέχθηκε η εφαρμογή. Από το header του request, αντλούμε το authorization. Ελέγχουμε αρχικά ότι αυτό υπάρχει και ότι αφορά το Bearer token. Εάν δεν ισχύουν οι συνθήκες αυτές επιστρέφουμε **false**. Αλλιώς, παίρνουμε το email που υπάρχει σαν πληροφορία στο token και , εφόσον υπάρχει, φορτώνουμε τον χρήστη με αυτό το email σε ένα UserDetails αντικείμενο. Εάν το token που δόθηκε υπάρχει, είναι έγκυρο και ανήκει σε υπαρκτό χρήστη, επιστρέφεται **true**, αλλιώς επιστρέφεται **false**.

```
public Boolean checkIfLogged(HttpServletRequest request) {
    final String authHeader = request.getHeader(AUTHORIZATION);
    final String userEmail;
    final String jwt;
    final String validHeaderPrefix = "Bearer ";

    if (authHeader == null || !authHeader.startsWith(validHeaderPrefix)) {
        log.info( msg: "No Authentication header was present");
        return false;
    }
    jwt = authHeader.substring(validHeaderPrefix.length());
    userEmail = jwtService.extractUsername(jwt);

    if (userEmail != null) {

        UserDetails userDetails = this.userDetailsService.loadUserByUsername(userEmail);
        var isTokenValid = tokenRepository.findByToken(jwt) Optional<Token>
                .map(t -> !t.isExpired() && !t.isRevoked()) Optional<Boolean>
                .orElse( other: false);

        return jwtService.isTokenValid(jwt, userDetails) && isTokenValid;
    }

    return false;
}
```

16. checkIfLogged μέθοδος του Authentication Service

2.4.5. Logout Service

Στο **LogoutService** είναι υλοποίηση του LogoutHandler interface, το οποίο ανήκει στο Spring Security, και βρίσκουμε μία μόνο μέθοδο, την **logout**, η οποία θέτει ως μη έγκυρο το token, εάν βρίσκεται στο header του request, και αδειάζει το context της SecurityContextHolder, κλάσης του Spring Security υπεύθυνο για την διαχείριση των στοιχείων του συνδεδεμένου χρήστη.

```
@Service
@RequiredArgsConstructor
public class LogoutService implements LogoutHandler {

  private final TokenRepository tokenRepository;

  chrisleskos *
  @Override
  public void logout(
      HttpServletRequest request,
      HttpServletResponse response,
      Authentication authentication
  ) {
    final String authHeader = request.getHeader( s: "Authorization");
    final String jwt;
    final String validHeaderPrefix = "Bearer ";

    if (authHeader == null ||!authHeader.startsWith(validHeaderPrefix)) {
      return;
    }
    jwt = authHeader.substring(validHeaderPrefix.length());
    var storedToken = tokenRepository.findByToken(jwt)
        .orElse( other: null);
    if (storedToken != null) {
      storedToken.setExpired(true);
      storedToken.setRevoked(true);
      tokenRepository.save(storedToken);
      SecurityContextHolder.clearContext();
    }
  }
}
```

17. Logout Service

2.4.6. Authentication Controller

Στην AuthenticationController κλάση, θα βρούμε τα **endpoints** μέσω των οποίων ο χρήστης αλληλεπιδρά με την εφαρμογή, και καλεί την υλοποιημένη λογική των **services** που αναλύσαμε παραπάνω.

```
@CrossOrigin
@RestController
@RequestMapping("/vaccoon-api/auth")
@RequiredArgsConstructor
public class AuthenticationController {

    private final AuthenticationService authService;
    private final LogoutService logoutService;
```

18. Authentication Controller

Τα endpoints της βρίσκονται κάτω από το κεντρικό path *"/vaccon-api/auth"*. Έχουμε προσθέσει το **CrossOrigin** annotation έτσι ώστε το API να μπορεί να δέχεται requests από διαφορετικά host names από αυτό στο οποίο τρέχει το ίδιο.

Στο *POST "/register"* endpoint, καλείται η **register** μέθοδος, η οποία επιστρέφει ένα response με κωδικό 200 (OK) και body (json) ένα αντικείμενο AuthenticationResponse, το οποίο θα παραχθεί από την register μέθοδο του AuthenticationService.

```
@PostMapping("/register")
public ResponseEntity<AuthenticationResponse> register(@RequestBody RegisterRequest request) {
    return ResponseEntity.ok(authService.register(request));
}
```

19. register μέθοδος του Authentication Controller

Στο *POST "/authenticate"* endpoint, καλείται η **authenticate** μέθοδος, η οποία επιστρέφει ένα response με κωδικό OK και body ένα αντικείμενο AuthenticationResponse, το οποίο θα παραχθεί από την authenticate μέθοδο του AuthenticationService.

```java
@PostMapping("/authenticate")
public ResponseEntity<AuthenticationResponse> authenticate(@RequestBody AuthenticationRequest request) {
    return ResponseEntity.ok(authService.authenticate(request));
}
```

20. authenticate μέθοδος του Authentication Controller

Στο *POST "/logout"* endpoint, καλείται η **logout** μέθοδος, η οποία καλεί την logout μέθοδο του LogoutService και επιστρέφει ένα response με κωδικό OK και body ένα string "Logout".

```java
@PostMapping("/logout")
public ResponseEntity<String> logout(HttpServletRequest request, HttpServletResponse response) {
    logoutService.logout(request, response, authentication: null);
    return ResponseEntity.ok( body: "Logout");
}
```

21. logout μέθοδος του Authentication Controller

Στο *GET "/check"* endpoint, καλείται η **checkIfLogged** μέθοδος, η οποία επιστρέφει ένα response με κωδικό OK και body μία Boolean μεταβλητή, η οποία θα υπολογισθεί από την checkIfLogged μέθοδο του AuthenticationService.

```java
@GetMapping("/check")
public ResponseEntity<Boolean> checkIfLogged(HttpServletRequest request, HttpServletResponse response) {
    return ResponseEntity.ok(authService.checkIfLogged(request));
}
```

22. checkIfLogged μέθοδος του Authentication Controller

2.5. Property

Στον **Property** φάκελο βρίσκονται τα αρχεία υπεύθυνα για τις λειτουργίες που αφορούν τις ιδιοκτησίες του χρήστη. Οι κατηγορίες των ιδιοκτησιών είναι αυτοκίνητο, δωμάτιο και τραπέζι εστιατορίου.

2.5.1. Μοντέλο Property

Το Property model, είναι το Parent model των υπόλοιπων ιδιοκτησιών που θα αναλυθούν στη συνέχεια. Τα κοινά πεδία που θα κληρονομήσουν είναι το **propertyId**, **propertyName**, **price** και ένα αντικείμενο **user**. Με βάση τον ορισμό του αντικειμένου ορίζεται το πως θα σχηματιστούν οι πίνακες και οι σχέσεις μεταξύ τους στην βάση δεδομένων. Έτσι, έχουμε ορίσει πως ο συνδυασμός user-property name θα είναι μοναδικός μέσω του Unique Constraint annotation. Το JsonIgnore annotation στο πεδίο του user αφορά την επίλυση ενός προβλήματος αναδρομικής ανάγνωσης του [4].

```java
@Data
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
@Table(uniqueConstraints = { @UniqueConstraint(columnNames = { "propertyName", "user_id" }) })
@JsonIgnoreProperties("hibernateLazyInitializer")
public class Property {
    @Id
    @GeneratedValue
    private Integer propertyId;
    private String propertyName;
    private long price;

    @ToString.Exclude
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id")
    @JsonIgnore
    private User user;
}
```

23. Μοντέλο Property

2.5.2. Μοντέλο Car

Το **Car** model, επεκτείνει το Property model και έχει επιπλέον το πεδίο **carType** το οποίο είναι ένα Enum που δηλώνει τον τύπο του αυτοκινήτου.

```java
@Data
@EqualsAndHashCode(callSuper = true)
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table
public class Car extends Property{
    private CarType carType;
}
```

24. Μοντέλο Car

Οι διαθέσιμες επιλογές του **CarType** είναι **SUV**, **Station Wagon**, **Coupe** και **Hatchback**.

```java
public enum CarType {
    no usages
    SUV,
    no usages
    SW,
    no usages
    COUPE,
    no usages
    HATCHBACK
}
```

25. Car Type Enum

2.5.3. Μοντέλο Room

Το **Room** model, επεκτείνει το Property model και έχει επιπλέον τα πεδία **numOfBeds** και **floor** τα οποία δηλώνουν αντίστοιχα τον αριθμό των κρεβατιών του δωματίου και τον όροφο στον οποίο βρίσκεται.

```
@Data
@EqualsAndHashCode(callSuper = true)
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table
public class Room extends Property{
    private int numOfBeds;
    private int floor;
}
```

26. Μοντέλο Room

2.5.4. Μοντέλο Restaurant Table

Το **ReastaurantTable** model, επεκτείνει το Property model και έχει επιπλέον το πεδίο **numOfSeats** το οποίο δηλώνει των αριθμό από καρέκλες που υπάρχουν στο τραπέζι.

```
@Data
@EqualsAndHashCode(callSuper = true)
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table
public class RestaurantTable extends Property {
    private int numOfSeats;
}
```

27. Μοντέλο Reastaurant Table

2.5.5. Value Objects

Κατά την δημιουργία νέων properties, ο χρήστης καλείται να παρέχει τα απαραίτητα στοιχεία τα οποία δεν χρειάζεται να έχουν constraints, συσχετισμούς με άλλα tables και ρόλους. Για τον λόγο αυτό, ο χρήστης αντί για τα models που περιγράψαμε παραπάνω, περνάει ένα object με το ίδιο όνομα και τύπο πεδίων. Τα objects αυτά κρατάνε την τιμή του πεδίου χωρίς περαιτέρω πληροφορίες. Οι έλεγχοι μοναδικότητας και η δημιουργία σχέσεων με άλλους πίνακες/αντικείμενα γίνεται αργότερα. Οι κλάσεις των αντικειμένων αυτών είναι οι **RoomRequest**, **CarRequest** και **TableRequest**. Όλες αυτές επεκτείνουν την κλάση **PropertyRequest**.

```
@Data
@Component
public class PropertyRequest {
    private int property_id;
    private String property_name;
    private long price;
}
```

28. Κλάση Property Request

Θα δούμε ως ενδεικτικό παράδειγμα το RoomRequest.

```
@EqualsAndHashCode(callSuper = true)
@Data
@Component
public class RoomRequest extends PropertyRequest{
    private int num_of_beds;
    private int floor;
}
```

29. Κλάση Room Request

Η RoomRequest κλάση επεκτείνει την PropertyRequest και έχει επιπλέον τα πεδία **num_of_beds** και **floor**, αυτά δηλαδή που είχε και το Room model. Παρατηρούμε ότι αυτή η κλάση έχει γίνει annotated ως **Component** και όχι ως **Entity** και **Table**, καθώς δεν έχει άμεση απεικόνιση στην βάση δεδομένων.

2.5.6. Repositories

Στον φάκελο repositories, βρίσκουμε τρία αρχεία, ένα για κάθε τύπο property. Τα αρχεία αυτά είναι το **CarRepository**, **RoomRepository** και **TableRepository**. Έχουν

όλα παρόμοια λογική. Επεκτείνουν το JpaRepository και εκτελούν τις ίδιες μεθόδους για την εκτέλεση query στην βάση, με μόνη διαφορά το αντικείμενο στο οποίο αναφέρεται το καθένα.

Πιο συγκεκριμένα, βρίσκουμε τις λειτουργίες συλλογής και διαγραφής ενός property με δεδομένο το **propertyId** και τον **user**, συλλογής όλων των properties με δεδομένο τον **user** και επεξεργασίας στοιχείων του property.

```java
@Repository
public interface CarRepository extends JpaRepository<Car, Integer> {
    1 usage  chrisleskos
    Optional<Car> findCarByPropertyIdAndUser(int property_id, User user);
    2 usages  chrisleskos
    List<Car> findAllByUser(User user);

    1 usage  chrisleskos
    @Transactional
    @Modifying
    @Query("delete from Car c where c.id = :id and c.user.id = :userId")
    int deleteByIdAndUserId(@Param("id") int id, @Param("userId") int userId);

    1 usage  chrisleskos
    @Transactional
    @Modifying
    @Query("update Car c set c.propertyName = :propertyName, c.price = :price, c.carType = :car_type " +
            "where c.propertyId = :propertyId")
    void updateCar(@Param("propertyId") Integer propertyId, @Param("propertyName") String propertyName, @Param("price") long price, @Param("car_type") CarType carType);
}
```

30. Car Repository

```java
@Repository
public interface RoomRepository extends JpaRepository<Room, Integer> {
    1 usage  chrisleskos
    Optional<Room> findRoomByPropertyIdAndUser(int property_id, User user);
    2 usages  chrisleskos
    List<Room> findAllByUser(User user);

    1 usage  chrisleskos
    @Transactional
    @Modifying
    @Query("delete from Room r where r.id = :id and r.user.id = :userId")
    int deleteByIdAndUserId(@Param("id") int id, @Param("userId") int userId);

    1 usage  chrisleskos
    @Transactional
    @Modifying
    @Query("update Room r set r.propertyName = :propertyName, r.price = :price, r.numOfBeds = :numOfBeds, r.floor = :floor " +
            "where r.propertyId = :propertyId")
    void updateRoom(@Param("propertyId") Integer propertyId, @Param("propertyName") String propertyName, @Param("price") long price, @Param("numOfBeds") int numOfBeds, @Param("
}
```

31. Room Repository

```java
@Repository
public interface TableRepository extends JpaRepository<RestaurantTable, Integer> {
    1 usage  chrisleskos
    Optional<RestaurantTable> findRestaurantTableByPropertyIdAndUser(int property_id, User user);
    2 usages  chrisleskos
    List<RestaurantTable> findAllByUser(User user);

    1 usage  chrisleskos
    @Transactional
    @Modifying
    @Query("delete from RestaurantTable rt where rt.id = :id and rt.user.id = :userId")
    int deleteByIdAndUserId(@Param("id") int id, @Param("userId") int userId);

    1 usage  chrisleskos
    @Transactional
    @Modifying
    @Query("update RestaurantTable t set t.propertyName = :propertyName, t.price = :price, t.numOfSeats = :numOfSeats " +
            "where t.propertyId = :propertyId")
    void updateTable(@Param("propertyId") int propertyId, @Param("propertyName") String propertyName, @Param("price") long price, @Param("numOfSeats") int numOfSeats);
}
```

32. Table Repository

2.5.7. Services

2.5.7.1. Car, Room & Table Services

Παρόμοια με τα repositories, έτσι και στα services των τριών properties, ακολουθείται κοινή γραμμή στην λογική τους, με μόνη διαφορά το property στο οποίο αναφερόμαστε. Στα services αυτά θα βρούμε την συλλογή όλων των properties, την συλλογή, αποθήκευση, διαγραφή και επεξεργασία ενός συγκεκριμένου property δεδομένου ενός id και του χρήστη.

Θα δούμε ενδεικτικά το παράδειγμα του **CarService**.

```
@Service
public class CarService {
    6 usages
    @Autowired
    private CarRepository repo;
    1 usage
    @Autowired
    private ReservationService reservationService;

    2 usages  chrisleskos
    public List<Car> getAllCars(LocalDate fromDate, LocalDate toDate, User user) {
        if (fromDate == null || toDate == null) return repo.findAllByUser(user);

        List<Reservation> reservations = reservationService.getReservations(fromDate, toDate, user);
        List<Car> cars = repo.findAllByUser(user);

        // Collect cars that their id is not matched with any of the properties id contained in reservations list
        return cars.stream()
                .filter(car -> reservations.stream()
                        .noneMatch(reservation -> reservation.getProperty().getPropertyId().equals(car.getPropertyId())))
                .toList();
    }

    3 usages  chrisleskos
    public Car getCar(int property_id, User user) throws PropertyNotFoundException {
        Optional<Car> car = repo.findCarByPropertyIdAndUser(property_id, user);
        if (car.isPresent()) return car.get();
        else throw new PropertyNotFoundException();
    }

    1 usage  chrisleskos
    public void saveCar(Car car) { repo.save(car); }

    1 usage  chrisleskos
    public int deleteCar(Car car) { return repo.deleteByIdAndUserId(car.getPropertyId(), car.getUser().getId()); }

    1 usage  chrisleskos *
    public void updateCar(Car car) {
        repo.updateCar(car.getPropertyId(), car.getPropertyName(), car.getPrice(), car.getCarType());
    }
}
```

33. Car Service

Στην μέθοδο **getAllCars**, περνάμε σαν παράμετρο μία αρχική ημερομηνία (**fromDate**), μία τελική ημερομηνία (**toDate**) και έναν χρήστη. Εάν δεν δοθεί κάποια από τις δύο ημερομηνίες, επιστρέφεται απλά η συλλογή όλων των αμαξιών του χρήστη από το **carRepository**. Εάν όμως δοθούν, συλλέγουμε όλα τα αμάξια του χρήστη, όλες τις κρατήσεις σε αυτό το διάστημα και φιλτράρουμε από την πρώτη

λίστα όσα αμάξια δεν βρίσκονται σε κάποια κράτηση της δεύτερης λίστας. Έτσι παίρνουμε την λίστα με όλα τα διαθέσιμα αμάξια σε μία δοσμένη περίοδο.

Η μέθοδος **getCar**, δέχεται σαν παράμετρο το **id** του property και τον **χρήστη**. Από τον **carRepository**, συλλέγει το αυτοκίνητο που αντιστοιχίζεται στα δύο αυτά στοιχεία και το επιστρέφει. Εάν δεν βρεθεί, πετάει ένα custom Exception (PropertyNotFoundException).

Στη μέθοδο **saveCar**, δίνουμε ένα αντικείμενο Car το οποίο αποθηκεύουμε με την **save** μέθοδο του **carRepository**.

Στη μέθοδο **deleteCar**, δίνουμε ένα αντικείμενο Car το οποίο διαγράφουμε με την **deleteByIdAndUserId** μέθοδο του **carRepository**.

Στη μέθοδο **updateCar**, δίνουμε ένα αντικείμενο Car το οποίο ενημερώνουμε τα στοιχεία του στην βάση με την **updateCar** μέθοδο του **carRepository**.

2.5.7.2. Property Service

Στο αρχείο **PropertyService** βρίσκουμε μία μέθοδο, την **getProperty**, η οποία παίρνει σαν παραμέτρους το **property id** και τον **χρήστη** της εφαρμογής. Η χρήση της μεθόδου αυτής είναι στο να ελέγχουμε ότι το property για το οποίο έγινε καινούργια κράτηση, ή επεξεργάστηκε κάποια υπάρχουσα, υπάρχει στην βάση. Ψάχνει λοιπόν στα τρία services που περιγράψαμε προηγουμένως, αν υπάρχει το property αυτό. Αν δεν το βρει σε κανένα, πετάει **PropertyNotFoundException**. Περισσότερα για τις κρατήσεις θα αναλυθούν παρακάτω.

```java
@Service
public class PropertyService {
    1 usage
    @Autowired
    RoomService roomService;
    1 usage
    @Autowired
    CarService carService;
    1 usage
    @Autowired
    TableService tableService;

    2 usages  chrisleskos
    public Property getProperty(int property_id, User user) throws PropertyNotFoundException {
        try {
            return roomService.getRoom(property_id, user);
        } catch (PropertyNotFoundException ignored) {}

        try {
            return carService.getCar(property_id, user);
        } catch (PropertyNotFoundException ignored) {}

        // Do not catch the last one, let it be handled
        return tableService.getTable(property_id, user);


    }
}
```

34. Property Service

2.5.8. Controllers

Οι Controllers των properties ακολουθούν επίσης μία κοινή λογική όπως έχουμε δει και σε προηγούμενα μέρη τις εφαρμογής. Καθώς το μόνο που αλλάζει είναι η ιδιοκτησία στην οποία αναφερόμαστε, θα χρησιμοποιηθεί μία από αυτές για να περιγραφεί η λειτουργία τους.

Στην κλάση **CarController**, εξυπηρετούμε κλήσεις της εφαρμογής κάτω από το endpoint *"/vacoon-api/car"*. Κάνουμε **Autowire** ένα Car Service αντικείμενο για να χρησιμοποιήσουμε την λογική που αναλύσαμε.

```java
@CrossOrigin
@RestController
@RequestMapping("/vaccoon-api/car")
public class CarController {
    6 usages
    @Autowired
    private CarService service;
```

35. Car Controller

Η πρώτη μέθοδος που συναντάμε είναι η **getAllCars**, όπου εξυπηρετεί **GET** requests στο endpoint *"vaccoon-api/car/"*, η οποία επιστρέφει ένα Response Entity με body μια λίστα από αντικείμενα Car. Οι παράμετροι οι οποίοι δέχεται η μέθοδος είναι δύο ημερομηνίες, **from** και **to**, οι οποίες όμως μπορούν και να παραληφθούν (null αν δεν έχουν δοθεί). Πρώτα παίρνουμε τον χρήστη από τα στοιχεία που βρίσκουμε στο SecurityContextHolder, το session δηλαδή του συνδεδεμένου χρήστη. Στην συνέχεια καλείται η getAllCars μέθοδος της CarService, με παραμέτρους τις ημερομηνίες και τον συνδεδεμένο χρήστη και φορτώνουμε σε μια λίστα από Car αντικείμενα το αποτέλεσμά της. Τέλος επιστρέφουμε ένα ResponseEntity με response κωδικό **OK** (200) και την λίστα που δημιουργήθηκε στο προηγούμενο βήμα στο body.

```java
@GetMapping({"/", ""})
public ResponseEntity<List<Car>> getAllCars(@RequestParam(value = "from", required = false) LocalDate fromDate,
                                            @RequestParam(value = "to", required = false) LocalDate toDate) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    User signedUser = (User) authentication.getPrincipal();
    List<Car> cars = service.getAllCars(fromDate, toDate, signedUser);
    return ResponseEntity.ok(cars);
}
```

36. getAllCars μέθοδος του Car Controller

Η μέθοδος **getCar**, καλείται στις **GET** κλήσεις κάτω από το endpoint *"vaccoon-api/car/{car-id}"*. Το **car-id** είναι ένα path variable το οποίο περνάει σαν παράμετρος στην μέθοδο. Πρώτα παίρνουμε τον χρήστη από τα στοιχεία που βρίσκουμε στο SecurityContextHolder, το session δηλαδή του συνδεδεμένου χρήστη. Στην συνέχεια καλείται η getCar μέθοδος της CarService, με παραμέτρους το car id και τον χρήστη. Εάν δεν βρεθεί κάποια τέτοια εγγραφή στη βάση, όπως αναφέραμε και στα services, παίρνουμε ένα PropertyNotFoundException το οποίο διαχειριζόμαστε επιστρέφοντας ένα ResponseEntity με response code **No Content** (204). Αλλιώς επιστρέφουμε **OK** (200) με το αυτοκίνητο που βρέθηκε.

```
@GetMapping("/{car_id}")
public ResponseEntity<Car> getCar(@PathVariable("car_id") int car_id) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    User signedUser = (User) authentication.getPrincipal();
    Car car;
    try {
        car = service.getCar(car_id, signedUser);
    } catch (PropertyNotFoundException e) {
        return ResponseEntity.noContent().build();
    }
    return ResponseEntity.ok(car);
}
```

37. getCar μέθοδος του Car Controller

Η μέθοδος **addProperty**, καλείται στις **POST** κλήσεις κάτω από το endpoint *"vaccoon-api/car/"*. Σαν παράμετρο περιμένει ένα JSON αντικείμενο το οποίο ταιριάζει με τη μορφή της **CarRequest** κλάσης. Πρώτα παίρνουμε τον χρήστη από το SecurityContextHolder. Στην συνέχεια δημιουργούμε ένα Car αντικείμενο από τα στοιχεία του CarReqeust και τον χρήστη και το περνάμε στην μέθοδο saveCar του CarService για να προστεθεί στο σύστημα.

```
@PostMapping
public void addProperty(@RequestBody @NotNull CarRequest carReq) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    User signedUser = (User) authentication.getPrincipal();

    Car car = Car.builder() CarBuilder<capture of ?, capture of ?>
            .propertyName(carReq.getProperty_name()) capture of ?
            .price(carReq.getPrice())
            .user(signedUser)
            .carType(carReq.getCarType())
            .build();

    service.saveCar(car);
}
```

38. addProperty μέθοδος του Car Controller

Η μέθοδος **editProperty**, καλείται στα **PUT** requests κάτω από το endpoint *"vaccoon-api/car/{car-id}"*. Το **car-id** είναι ένα path variable το οποίο περνάει σαν παράμετρος στην μέθοδο. Επίσης η μέθοδος περιμένει ένα JSON αντικείμενο στη μορφή του CarRequest. Πρώτα παίρνουμε τον χρήστη από το SecurityContextHolder. Στην συνέχεια καλείται η getCar μέθοδος της CarService, με παραμέτρους το car id και τον χρήστη και κρατάμε σε ένα αντικείμενο την εγγραφή που αντιστοιχούν. Εάν δεν βρεθεί κάποια τέτοια εγγραφή στη βάση, όπως αναφέραμε και στα services, παίρνουμε ένα PropertyNotFoundException το οποίο διαχειριζόμαστε επιστρέφοντας ένα ResponseEntity με response code **No Content** (204). Αλλιώς, ενημερώνουμε τις τιμές του αντικειμένου σύμφωνα με τις τιμές του CarRequest που έχει δοθεί και επιστρέφουμε **OK** (200) με μία true Boolean μεταβλητή.

```
@PutMapping("/{car_id}")
public ResponseEntity<Boolean> editProperty(@PathVariable("car_id") int car_id, @RequestBody @NotNull CarRequest carReq) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    User signedUser = (User) authentication.getPrincipal();
    Car car;
    try {
        car = service.getCar(car_id, signedUser);
        car.setPrice(carReq.getPrice());
        car.setPropertyName(carReq.getProperty_name());
        car.setCarType(carReq.getCarType());
        service.updateCar(car);

        return ResponseEntity.ok( body: true);
    } catch (PropertyNotFoundException e) {
        return ResponseEntity.noContent().build();
    }
}
```

39. editProperty μέθοδος του Car Controller

Η μέθοδος **deleteProperty**, καλείται στα **DELETE** requests κάτω από το endpoint *"vaccoon-api/car/{car-id}"*. Το **car-id** είναι ένα path variable το οποίο περνάει σαν παράμετρος στην μέθοδο. Πρώτα παίρνουμε τον χρήστη από το SecurityContextHolder. Στην συνέχεια φτιάχνεται ένα νέο Car αντικείμενο μόνο με το car id και τον χρήστη το οποίο περνάει σαν παράμετρος στη deleteCar μέθοδο της CarService. Επιστρέφουμε **OK** (200) με μία Boolean μεταβλητή, η τιμή της οποίας ορίζεται από τον εάν διαγράφηκε τουλάχιστον μία εγγραφή στον πίνακα.

```
@DeleteMapping("/{car_id}")
public ResponseEntity<Boolean> deleteProperty(@PathVariable("car_id") int car_id) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    User signedUser = (User) authentication.getPrincipal();

    Car car = Car.builder() CarBuilder<capture of ?, capture of ?>
            .propertyId(car_id) capture of ?
            .user(signedUser)
            .build();

    return ResponseEntity.ok( body: service.deleteCar(car) > 0);
}
```

40. deleteProperty μέθοδος του Car Controller

2.6. Client

2.6.1. Μοντέλο Client

Το μοντέλο **Client**, περιέχει τα απαραίτητα στοιχεία ενός πελάτη. Τα στοιχεία αυτά είναι το clientId, το όνομα, το επίθετο, ο αριθμός τηλεφώνου, το email, το έτος γέννησης, η χώρα καταγωγής, το φύλο, σχετικά σχόλια και η εντύπωση που έχει αφήσει. Επίσης έχει το πεδίο user το οποίο αναφέρει σε ποιόν χρήστη ανήκει ο πελάτης αυτός. Το email και ο αριθμός τηλεφώνου, έχουν unique constraints, δηλαδή, δεν μπορούν να υπάρχουν διπλότυπα σε διαφορετικές εγγραφές του πίνακα.

```java
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(uniqueConstraints = { @UniqueConstraint(columnNames = { "phoneNum", "user_id" }),  @UniqueConstraint(columnNames = { "email", "user_id" }) })
@JsonIgnoreProperties("hibernateLazyInitializer")
public class Client {
    @Id
    @GeneratedValue
    private int clientId;
    private String firstname;
    private String lastname;
    private String phoneNum;
    private String email;
    private int yearOfBirth;
    private Locale country;
    private ClientGender gender;
    private String comments;
    private int impression;

    @ToString.Exclude
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id")
    @JsonIgnore
    private User user;
}
```

41. Μοντέλο Client

2.6.2. Client Value Object

Το **ClientReq** περιέχει τα ίδια πεδία με το μοντέλο, εκτός του χρήστη, και θα χρησιμοποιηθεί για να μπορεί ο χρήστης της εφαρμογής να στέλνει στοιχεία για δημιουργία ή ενημέρωση κάποιας εγγραφής χρήστη χωρίς περαιτέρω πληροφορία από τα δεδομένα αυτά καθαυτά.

```java
@Data
@Component
public class ClientReq {
    private int clientId;
    private String firstname;
    private String lastname;
    private String phoneNum;
    private String email;
    private int yearOfBirth;
    private Locale country;
    private ClientGender gender;
    private String comments;
    private int impression;
}
```

42. Κλάση ClientReq

2.6.3. Client Repository

Στο **ClientRepository**, κάνουμε extend το **JpaRepository** και βρίσκουμε τις μεθόδους **findAllByUser** για την συλλογή όλων των πελατών ενός χρήστη, **findClientByClientIdAndUser** για την εύρεση ενός πελάτη που ανήκει σε έναν χρήστη με συγκεκριμένο client id, **findByEmailAndUser** για την εύρεση του πελάτη ενός χρήστη με συγκεκριμένο email, **findByPhoneAndUser** για την εύρεση του πελάτη ενός χρήστη με συγκεκριμένο τηλέφωνο, **deleteByIdAndUserId** για την διαγραφή του πελάτη ενός χρήστη με συγκεκριμένο client id, **update** για την ενημέρωση των στοιχείων του πελάτη ενός χρήστη.

```
@Repository
public interface ClientRepository extends JpaRepository<Client, Integer> {
    1 usage  chrisleskos
    List<Client> findAllByUser(User user);

    1 usage  chrisleskos
    Optional<Client> findClientByClientIdAndUser(int client_id, User user);

    2 usages  chrisleskos
    @Query("select c from Client c where c.email = :email and c.user = :user")
    Optional<Client> findByEmailAndUser(@Param("email") String email, @Param("user") User user);

    no usages  chrisleskos
    @Query("select c from Client c where c.phoneNum = :phoneNum and c.user = :user")
    Optional<Client> findByPhoneNumAndUser(@Param("phoneNum") String phoneNum, @Param("user") User user);

    1 usage  chrisleskos
    @Transactional
    @Modifying
    @Query("delete from Client c where c.clientId = :id and c.user.id = :userId")
    int deleteByIdAndUserId(@Param("id") int id, @Param("userId") int userId);

    chrisleskos
    @Transactional
    @Modifying
    @Query("update Client c set c.firstname = :firstname, c.lastname = :lastname, c.email = :email, c.phoneNum = :phoneNum " + "where c.clientId = :clientId")
    void update(@Param("clientId") int clientId, @Param("firstname") String firstname, @Param("lastname") String lastname, @Param("email") String email, @Param("phoneNum") St
}
```

43. Client Repository

2.6.4. Client Service

Η μέθοδος **getAllClients** δέχεται ένα αντικείμενο User και επιστρέφει το αποτέλεσμα της findAllByUser μεθόδου του ClientRepository.

```
public List<Client> getAllClients(User signedUser) {
    return repository.findAllByUser(signedUser);
}
```

44. getAll Clients μέθοδος του Client Service

Η μέθοδος **getClient** δέχεται σαν παραμέτρους ένα client id και έναν χρήστη. Στη συνέχεια καλεί την findByClientIdAndUser μέθοδο του ClientRepository. Αν βρεθεί πελάτης με αυτά τα στοιχεία, επιστρέφει το αντικείμενο αυτό, αλλιώς πετάει ClientNotFoundException, μια custom exception της εφαρμογής.

```
public Client getClient(int clientId, User signedUser) throws ClientNotFoundException {
    Optional<Client> client = repository.findClientByClientIdAndUser(clientId, signedUser);
    if (client.isPresent()) return client.get();
    else throw new ClientNotFoundException();
}
```

45. getClient μέθοδος του Client Service

Η μέθοδος getClientByEmail, δέχεται ένα email και έναν χρήστη και καλεί την findByEmailAndUser μέθοδο του ClientRepository. Αν βρεθεί πελάτης με αυτά τα στοιχεία, επιστρέφει το αντικείμενο αυτό, αλλιώς πετάει ClientNotFoundException.

```
public Client getClientByEmail(String email, User signedUser) throws ClientNotFoundException {
    Optional<Client> client = repository.findByEmailAndUser(email, signedUser);
    if (client.isPresent()) return client.get();
    else throw new ClientNotFoundException();
}
```

46. getClientByEmail μέθοδος του Client Service

Η μέθοδος **getClientByPhoneNum**, δέχεται ένα email και έναν χρήστη και καλεί την findByPhoneNumAndUser μέθοδο του ClientRepository. Αν βρεθεί πελάτης με αυτά τα στοιχεία, επιστρέφει το αντικείμενο αυτό, αλλιώς πετάει ClientNotFoundException.

```
public Client getClientByPhoneNum(String phoneNum, User signedUser) throws ClientNotFoundException {
    Optional<Client> client = repository.findByPhoneNumAndUser(phoneNum, signedUser);
    if (client.isPresent()) return client.get();
    else throw new ClientNotFoundException();
}
```

47. getClientByPhoneNum μέθοδος του Client Service

Η μέθοδος **addClient**, δέχεται ένα αντικείμενο Client το οποίο και αποθηκεύει με τη save μέθοδο του ClientRepository.

```
public void addClient(Client client) {
    repository.save(client);
}
```

48. addClient μέθοδος του Client Service

Η μέθοδος **updateClient**, δέχεται ένα αντικείμενο Client και από τα στοιχεία του συμπληρώνει τις παραμέτρους της update μεθόδου του ClientRepository.

```
public void updateClient(Client client) {
    repository.update(client.getClientId(), client.getFirstname(), client.getLastname(), client.getEmail(), client.getPhoneNum());
}
```

49. updatelient μέθοδος του Client Service

Η μέθοδος **deleteClient**, δέχεται ένα αντικείμενο Client και από τα στοιχεία του περνάει το client id και το user id σαν παραμέτρους στην μέθοδο deleteByIdAndUserId του ClientRepository.

```
public int deleteClient(Client client) {
    return repository.deleteByIdAndUserId(client.getClientId(), client.getUser().getId());
}
```

50. deleteClient μέθοδος του Client Service

2.6.5. Client Controller

Στην κλάση **ClientController**, εξυπηρετούμε κλήσεις της εφαρμογής κάτω από το endpoint *"/vacoon-api/client"*. Κάνουμε **Autowire** ένα ClientService αντικείμενο για να χρησιμοποιήσουμε την λογική που αναλύσαμε.

```
@CrossOrigin
@RestController
@RequestMapping("/vaccoon-api/client")
public class ClientController {
    6 usages
    @Autowired
    private ClientService service;
```

51. Client Controller

Η πρώτη μέθοδος που συναντάμε είναι η **getAllClients**, όπου εξυπηρετεί **GET** requests στο endpoint *"/vaccoon-api/client/"*, η οποία επιστρέφει ένα Response Entity με body μια λίστα από αντικείμενα Client. Πρώτα παίρνουμε τον χρήστη από τα στοιχεία που βρίσκουμε στο SecurityContextHolder, το session δηλαδή του συνδεδεμένου χρήστη. Στην συνέχεια καλείται η getAllClients μέθοδος της ClientService, με παραμέτρους τις ημερομηνίες και τον συνδεδεμένο χρήστη και φορτώνουμε σε μια λίστα από Client αντικείμενα το αποτέλεσμά της. Τέλος επιστρέφουμε ένα ResponseEntity με response κωδικό **OK** (200) και την λίστα που δημιουργήθηκε στο προηγούμενο βήμα στο body.

```
@GetMapping({"/", ""})
public ResponseEntity<List<Client>> getAllClients() {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    User signedUser = (User) authentication.getPrincipal();
    List<Client> clients = service.getAllClients(signedUser);
    return ResponseEntity.ok(clients);
}
```

52. getAllClient μέθοδος του Client Controller

Η μέθοδος **getClient**, εκτελείται όταν δέχεται η εφαρμογή **GET** request στο endpoint *"vaccoon-api/client/{id}"* , όπου id είναι το αναγνωριστικό του πελάτη που θα αναζητήσουμε. Παίρνουμε πρώτα των συνδεδεμένο χρήστη από το SecurityContextHolder και καλούμε την getClient μέθοδο του ClientService. Εάν ο πελάτης βρεθεί, επιστρέφουμε ένα Response Entity με response code **OK** και στο body τον πελάτη που βρέθηκε. Εάν δεν βρεθεί, επιστρέφει ένα Response Entity με κωδικό **No Content**.

```
@GetMapping("/{client_id}")
public ResponseEntity<Client> getClient(@PathVariable("client_id") int client_id) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    User signedUser = (User) authentication.getPrincipal();
    Client client;
    try {
        client = service.getClient(client_id, signedUser);
        return ResponseEntity.ok(client);
    } catch (ClientNotFoundException e) {
        return ResponseEntity.noContent().build();
    }
}
```

53. getClient μέθοδος του Client Controller

Η μέθοδος **saveClient**, καλείται στις **POST** κλήσεις κάτω από το endpoint *"vaccoon-api/client/"*. Σαν παράμετρο περιμένει ένα JSON αντικείμενο το οποίο ταιριάζει με τη μορφή της **ClientReq** κλάσης. Πρώτα παίρνουμε τον χρήστη από το SecurityContextHolder. Στην συνέχεια δημιουργούμε ένα Client αντικείμενο από τα στοιχεία του ClientReq και τον χρήστη και το περνάμε στην μέθοδο addClient του ClientService για να προστεθεί στο σύστημα.

```
@PostMapping
public void saveClient(@RequestBody @NotNull ClientReq clientReq) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    User signedUser = (User) authentication.getPrincipal();
    Client client = Client.builder()
            .firstname(clientReq.getFirstname())
            .lastname(clientReq.getLastname())
            .email(clientReq.getEmail())
            .phoneNum(clientReq.getPhoneNum())
            .yearOfBirth(clientReq.getYearOfBirth())
            .country(clientReq.getCountry())
            .gender(clientReq.getGender())
            .comments(clientReq.getComments())
            .impression(clientReq.getClientId())
            .user(signedUser)
            .build();

    service.addClient(client);
}
```

54. saveClient μέθοδος του Client Controller

Η μέθοδος **updateClient**, καλείται στα **PUT** requests κάτω από το endpoint *"vaccoon-api/client/{client-id}"*. Το **client-id** είναι ένα path variable το οποίο περνάει σαν παράμετρος στην μέθοδο. Επίσης η μέθοδος περιμένει ένα JSON αντικείμενο στη μορφή του ClientReq. Πρώτα παίρνουμε τον χρήστη από το SecurityContextHolder. Στην συνέχεια καλείται η getClient μέθοδος της ClientService, με παραμέτρους το client id και τον χρήστη και κρατάμε σε ένα αντικείμενο την εγγραφή που αντιστοιχούν. Εάν δεν βρεθεί κάποια τέτοια εγγραφή στη βάση, όπως αναφέραμε και στα services, παίρνουμε ένα ClientNotFoundException το οποίο διαχειριζόμαστε επιστρέφοντας ένα ResponseEntity με response code **No Content** (204). Αλλιώς, ενημερώνουμε τις τιμές του αντικειμένου σύμφωνα με τις τιμές του ClientReq που έχει δοθεί και επιστρέφουμε **OK** (200) με μία true Boolean μεταβλητή.

```
@PutMapping("/{client_id}")
public ResponseEntity<Boolean> updateClient(@RequestBody @NotNull ClientReq clientReq, @PathVariable("client_id") int client_id) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    User signedUser = (User) authentication.getPrincipal();
    try {
        Client client = service.getClient(client_id, signedUser);
        client.setFirstname(clientReq.getFirstname());
        client.setLastname(clientReq.getLastname());
        client.setEmail(clientReq.getEmail());
        client.setPhoneNum(clientReq.getPhoneNum());
        client.setYearOfBirth(clientReq.getYearOfBirth());
        client.setCountry(clientReq.getCountry());
        client.setGender(clientReq.getGender());
        client.setComments(clientReq.getComments());
        client.setImpression(clientReq.getImpression());

        service.updateClient(client);

        return ResponseEntity.ok( body: true);
    } catch (ClientNotFoundException e) {
        return ResponseEntity.noContent().build();
    }
}
```

55. updateClient μέθοδος του Client Controller

Η μέθοδος **deleteClient**, καλείται στα **DELETE** requests κάτω από το endpoint *"vaccoon-api/client/{client-id}"*. Το **client-id** είναι ένα path variable το οποίο περνάει σαν παράμετρος στην μέθοδο. Πρώτα παίρνουμε τον χρήστη από το SecurityContextHolder. Στην συνέχεια φτιάχνεται ένα νέο Client αντικείμενο μόνο με το client id και τον χρήστη το οποίο περνάει σαν παράμετρος στη deleteClient μέθοδο της ClientService. Επιστρέφουμε **OK** (200) με μία Boolean μεταβλητή, η τιμή της οποίας ορίζεται από τον εάν διαγράφηκε τουλάχιστον μία εγγραφή στον πίνακα.

```
@DeleteMapping("/{client_id}")
public ResponseEntity<Boolean> deleteClient(@PathVariable("client_id") int client_id) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    User signedUser = (User) authentication.getPrincipal();
    Client client = Client.builder()
            .clientId(client_id)
            .user(signedUser)
            .build();

    return ResponseEntity.ok( body: service.deleteClient(client) > 0);
}
```

56. deleteClient μέθοδος του Client Controller

2.7. Reservation

2.7.1. Μοντέλο Reservation

Στα στοιχεία της κλάσης Reservation βρίσκουμε το **reservation id** όπου είναι το αναγνωριστικό της κράτησης, το **reservation date**, δηλαδή την ημερομηνία και ώρα που έγινε η κράτηση, **number of people** για τον αριθμό των ανθρώπων της κράτησης, ένα **Property** αντικείμενο, μία μεταβλητή **propertyType** του τύπου δηλαδή του Property (Car/Room/Restaurant Table) , το **status** το οποίο μπορεί να είναι σε αναμονή, ακυρωμένη, ελεγμένη ή ολοκληρωμένη, ένα αντικείμενο **Client** που αφορά αυτόν που έκανε την κράτηση, τον **User** της εφαρμογής, τις ημερομηνίες **από** και **έως** της κράτησης, ημερομηνία και ώρα **ελέγχου** και **ολοκλήρωσης** (αν έχουν γίνει), και **συνολική τιμή** για το συγκεκριμένο property και τις ημερομηνίες.

```java
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table
public class Reservation {
    @Id
    @GeneratedValue
    private int reservationId;
    private LocalDateTime reservationDate;
    private int numOfPeople;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "property_id")
    private Property property;

    private ReservationStatus status;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "client_id")
    private Client client;

    @ToString.Exclude
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id")
    @JsonIgnore
    private User user;

    private LocalDate fromDate;
    private LocalDate toDate;
    private LocalDateTime checkIn;
    private LocalDateTime checkOut;
    private float totalPrice;
    private String propertyType;
}
```

57. Μοντέλο Reservation

2.7.2. Reservation Value Object

Όπως ήδη έχουμε δει και σε προηγούμενα μοντέλα, χρησιμοποιούμε ένα *value object*, το **ReservationReq** στην προκειμένη περίπτωση, ως μια αντιγραφή των πεδίων του βασικού μοντέλου (Reservation) χωρίς περαιτέρω πληροφορία που αφορά την σχεσιακή αποθήκευση και συλλογή των δεδομένων αλλά και στοιχείων που δεν παρέχει ο χρήστης αλλά συμπληρώνει το σύστημα μόνο του. Παρατηρούμε εδώ ότι και το αντικείμενο του πελάτη έχει αντικατασταθεί από το αντίστοιχο value object του Client (**ClientReq**).

```
@Data
@Component
public class ReservationReq {
    private int reservationId;
    private int propertyId;
    private ClientReq clientreq;
    private int numOfPeople;
    private LocalDate fromDate;
    private LocalDate toDate;
    private LocalDateTime checkIn;
    private LocalDateTime checkOut;
    private float totalPrice;
    private String propertyType;
    private ReservationStatus status;
}
```

58. Κλάση Reservation Request

2.7.3. Reservation Repository

Το **ReservationRepository** interface κάνει extend το JpaRepository. Έτσι έχουμε τα ίδια προνόμια που έχουμε αναλύσει στα άλλα repositories.

Στην μέθοδο **findAllByUser** δίνουμε ένα αντικείμενο User και βρίσκουμε όλες τις κρατήσεις που αντιστοιχούν στον χρήστη αυτόν.

```
@Query("select r from Reservation r where r.user = :user order by r.reservationDate desc")
List<Reservation> findAllByUser(@Param("user") User user);
```

59. Reservation Repository

Στην μέθοδο **findValidReservationByPeriod**, περνάμε ημερομηνία από, ημερομηνία έως, το canceled status από το enum με τις καταστάσεις της κράτησης και τον χρήστη της εφαρμογής. Με βάση αυτά, βρίσκουμε όλες τις μη ακυρωμένες κρατήσεις, που βρίσκονται μεταξύ της περιόδου που περικλείουν οι ημερομηνίες που δόθηκαν και ανήκουν στον συγκεκριμένο χρήστη.

```
@Query("select r from Reservation r where not (r.fromDate >= :toPeriodDate or r.toDate <= :fromPeriodDate) and r.status != :canc
List<Reservation> findValidReservationsByPeriod(@Param("fromPeriodDate") LocalDate fromDate, @Param("toPeriodDate") LocalDate to
```

60. findValidReservationsByPeriod μέθοδος του Reservation Repository

Παρόμοια λειτουργία έχει και η μέθοδος **findValidReservationsByDate**, με τη μόνη διαφορά πως δεν αναζητά περίοδο αλλά ακριβή αντιστοίχιση μόνο με το **from date** που το δίνουμε.

```
@Query("select r from Reservation r where r.fromDate = :fromDate and r.status != :canceledStatus and r.user = :
List<Reservation> findValidReservationsByDate(@Param("fromDate") LocalDate fromDate, @Param("canceledStatus") R
```

61. findValidReservationsByDate μέθοδος του Reservation Repository

Επίσης παρόμοια λειτουργία θα βρούμε στην μέθοδο **findReservationsByYearPeriodAndPropertyType**, όπου εδώ δεν περίοδο μεταξύ ολόκληρων ημερομηνιών, αλλά μεταξύ ετών και αντιστοιχίζονται τα αποτελέσματα σε ένα συγκεκριμένο **property type** που θα δοθεί.

```
@Query("select r from Reservation r where YEAR(r.fromDate) >= :fromYear and YEAR(r.fromDate) <= :toYear and
List<Reservation> findReservationsByYearPeriodAndPropertyType(@Param("fromYear") Integer fromYear, @Param("t
```

62. findReservationsByYearAndPropertyType μέθοδος του Reservation Repository

Η μέθοδος **findReservationByReservationIdAndUser**, μας επιστρέφει ένα αντικείμενο Reservation, εάν βρει αντιστοίχιση για συγκεκριμένο id και χρήστη.

```
Optional<Reservation> findReservationByReservationIdAndUser(int reservationId, User signedUser);
```

63. findReservationByReservationIdAndUser μέθοδος του Reservation Repository

Η **deleteByIdAndUserId** μέθοδος, διαγράφει μία εγγραφή με συγκεκριμένο id και χρήστη.

```
@Transactional
@Modifying
@Query("delete from Reservation r where r.reservationId = :id and r.user.id = :userId")
int deleteByIdAndUserId(@Param("id") int id, @Param("userId") int userId);
```

64. deleteByIdAndUserId μέθοδος του Reservation Repository

Τέλος, η **update** μέθοδος, ενημερώνει τα στοιχεία της κράτησης με τα νέα που δίνονται σαν παράμετροι

```
@Transactional
@Modifying
@Query("update Reservation r set r.reservationDate = :reservationDate, r.numOfPeople = :numOfPeople, r.proper
void update(@Param("reservationId") int reservationId, @Param("reservationDate") LocalDateTime reservationDat
```

65. update μέθοδος του Reservation Repository

2.7.4. Μοντέλο Reservations Statistics

Στην κλάση **ReservationStatistics**, βρίσκουμε πεδία από λίστες του αντικειμένου FreqDistribution, ένα πεδίο για τον συνολικό αριθμό των κρατήσεων και ένα για τα συνολικά έσοδα. Στόχος είναι να κρατάει τα απαραίτητα στοιχεία για τα στατιστικά νούμερα που παρέχει η εφαρμογή. Τα στατιστικά αφορούν τόσο συχνότητα όσο και κερδοφορία, για αυτόν τον λόγο βρίσκουμε και τα δύο τελευταία πεδία.

```
@Builder
@Data
@NoArgsConstructor
@AllArgsConstructor
public class ReservationsStatistics {
    private List<FreqDistribution> propertyFreqDistribution;
    private List<FreqDistribution> monthFreqDistribution;
    private List<FreqDistribution> yearFreqDistribution;
    private List<FreqDistribution> weekdayFreqDistribution;
    private List<FreqDistribution> genderFreqDistribution;
    private int totalReservations;
    private float totalIncome;
}
```

66. Μοντέλο Rservations Statistics

Το frequency distribution αφορά την κατανομή της συχνότητας ενός αντικειμένου/μέτρου. Στην εφαρμογή απεικονίζεται συγκρατώντας το αντικείμενο το οποίο εξετάζουμε, τον αριθμό των οποίο το βρίσκουμε μέσα σε κάποια κράτηση και τα συνολικά έσοδα που έχουν επιφέρει οι κρατήσεις αυτές.

```java
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Data
public class FreqDistribution {
    private String name;
    private int reservations;
    private float income;
}
```

67. Μοντέλο Frequency Distribution

Επιστρέφοντας έτσι στη ReservationsStatistics κλάση, οι λίστες των FreqDistribution αποτελούν την κατανομή συχνότητας και κερδοφορίας κοινών αντικειμένων. Πιο συγκεκριμένα, το **propertyFreqDistribution**, είναι η λίστα που περιέχει πολλά FreqDestribution αντικείμενα τα οποία, καθένα από αυτά αφορά ένα property. Το **monthFreqDistribution** είναι μία λίστα FreqDistribution αντικειμένων, όπου καθένα από αυτά αφορά έναν μήνα. Εύκολα μπορεί να κατανοηθεί και η λειτουργία των υπόλοιπων λιστών που αφορούν year (έτος), weekday (ημέρα της εβδομάδας) και gender (φύλο) αντίστοιχα.

2.7.5. Reservation Service

Στο **ReservationService** γίνεται **Autowire** το ReservationRepository για να χρησιμοποιήσουμε τις μεθόδους που αναλύσαμε παραπάνω.

```java
@Service
public class ReservationService {

    8 usages
    @Autowired
    private ReservationRepository reservationRepository;
```

68. Reservation Service

Στην **getReservations** μέθοδο, με παράμετρο μόνο ένα αντικείμενο χρήστη, επιστρέφουμε το αποτέλεσμα της findAllByUser μεθόδου του ReservationRepository.

```
private List<Reservation> getReservations(User user) {
    return reservationRepository.findAllByUser(user);
}
```

69. 1η υλοποίηση της getReservations μεθόδου του Reservation Service

Η μέθοδος **getReservations** γίνεται *overload*, ορίζεται δηλαδή διαφορετικά όταν δέχεται δύο παραμέτρους με ημερομηνίες from και to και έναν χρήστη. Εάν κάποια από τις ημερομηνίες είναι null, καλείται η περίπτωση της getReservations που δέχεται σαν παράμετρο μόνο τον χρήστη, και επιστρέφει όλες τις κρατήσεις χωρίς περιορισμό περιόδου. Εάν έχουν δοθεί και οι δύο, καλείται η findValidReservationsByPeriod μέθοδος του ReservationRepository.

```
public List<Reservation> getReservations(LocalDate fromDate, LocalDate toDate, User user) {
    if (fromDate == null || toDate == null) return getReservations(user);

    return reservationRepository.findValidReservationsByPeriod(fromDate, toDate, ReservationStatus.CANCELED, user);
}
```

70. 2η υλοποίηση της getReservations μεθόδου του Reservation Service

Μία τρίτη υλοποίηση της **getReservations** δέχεται μόνο μία ημερομηνία (fromDate) και τον χρήστη. Μέσα σε αυτή, καλείται η findValidReservationsByDate μέθοδος του ReservationRepository. Τη χρησιμότητα αυτής της μεθόδου την βρίσκουμε όταν θέλουμε να βρούμε διαθέσιμα τραπέζια, η διαθεσιμότητα των οποίων δεν αφορά περίοδο μεγαλύτερη από μία μέρα.

```
public List<Reservation> getReservations(LocalDate fromDate, User user) {
    return reservationRepository.findValidReservationsByDate(fromDate, ReservationStatus.CANCELED, user);
}
```

71. 3η υλοποίηση της getReservations μεθόδου του Reservation Service

Η **getReservationsFromYearPeriod** μέθοδος, δέχεται δύο ακέραιους ως αρχικό και τελικό έτος που αναζητούμε, τον τύπο του property και έναν χρήστη. Επιστρέφει το αποτέλεσμα της findReservationsByYearPeriodAndPropertyType του ReservationRepository.

```
public List<Reservation> getReservationsFromYearPeriod(int fromYear, int toYear, String propertyType, User user) {
    return reservationRepository.findReservationsByYearPeriodAndPropertyType(fromYear, toYear, propertyType, ReservationStatus.CANCELED, user)
}
```

72. getReservationsFromYearPeriod μέθοδος του Reservation Service

Η μέθοδος **getReservation**, παίρνει ένα reservation id και τον συνδεδεμένο χρήστη, και επιστρέφει, μέσω της findReservationByReservationIdAndUser μέθοδο του ReservationRepository, το αντικείμενο της κράτησης που αντιστοιχεί. Εάν δεν βρεθεί, δίνει ένα ReservationNotFoundException.

```
public Reservation getReservation(int reservation_id, User signedUser) throws ResevationNotFoundException {
    Optional<Reservation> reservation = reservationRepository.findReservationByReservationIdAndUser(reservation_id, signedUser);
    if (reservation.isPresent()) return reservation.get();
    else throw new ResevationNotFoundException();
}
```

73. getReservation μέθοδος του Reservation Service

Στη μέθοδο **saveReservation**, περνάμε ένα Reservation αντικείμενο και το αποθηκεύουμε μέσω του ReservationRepository.

```
public void saveReservation(Reservation reservation) {
    reservationRepository.save(reservation);
}
```

74. saveReservation μέθοδος του Reservation Service

Η **updateReservation** μέθοδος, δέχεται ένα αντικείμενο Reservation και περνάει τα δεδομένα του στην update μέθοδο του ReservationRepository, η οποία ενημερώνει την εγγραφή στη βάση.

```
public void updateReservation(Reservation reservation) {
    reservationRepository.update(reservation.getReservationId(),
            reservation.getReservationDate(),
            reservation.getNumOfPeople(),
            reservation.getProperty().getPropertyId(),
            reservation.getStatus(),
            reservation.getClient().getClientId(),
            reservation.getFromDate(),
            reservation.getToDate(),
            reservation.getCheckIn(),
            reservation.getCheckOut(),
            reservation.getTotalPrice(),
            reservation.getPropertyType());
}
```

75. updateReservation μέθοδος του Reservation Service

Η **deleteReservation** μέθοδος, δέχεται ένα αντικείμενο Reservation και περνάει τα reservation id και user id στη deleteByIdAndUserId μέθοδο του ReservationRepository, η οποία διαγράφει την εγγραφή στη βάση.

```
public int deleteReservation(Reservation reservation) {
    return reservationRepository.deleteByIdAndUserId(reservation.getReservationId(), reservation.getUser().getId());
}
```

76. deleteReservation μέθοδος του Reservation Service

Η τελευταία και μεγαλύτερη μέθοδος του ReservationService είναι η **getStats** η οποία δέχεται μία λίστα από κρατήσεις, μία λίστα από ιδιοκτησίες, έναν ακέραιο το αρχικού έτους και έναν του τελικού έτους της περιόδου που μας αφορά. Σε αυτήν δημιουργούμε ένα ReservationStatistics αντικείμενο. Όπως έχουμε δει, η ReservationStatistics κλάση περιέχει πέντε λίστες FreqDistribution αντικειμένων. Έτσι η getStats ξεκινάει αρχικοποιώντας πέντε λίστες, μία για κάθε αντίστοιχο πεδίο. Επίσης ορίζουμε και μία μεταβλητή **totalIncome** για να υπολογισθούν τα συνολικά έσοδα των κρατήσεων.

```
public ReservationsStatistics getStats(List<Reservation> reservations, List<Property> properties, Integer from, Integer to) {
    List<FreqDistribution> propertyFreqDistribution = new ArrayList<>();
    List<FreqDistribution> monthFreqDistribution = new ArrayList<>();
    List<FreqDistribution> yearFreqDistribution = new ArrayList<>();
    List<FreqDistribution> weekdayFreqDistribution = new ArrayList<>();
    List<FreqDistribution> genderFreqDistribution = new ArrayList<>();
    float totalIncome = 0;
```

77. getStats μέθοδος του Reservation Service

Στη συνέχεια προσθέτουμε στις λίστες τις αρχικές τιμές τους. Πιο συγκεκριμένα, αυτές θα είναι αντικείμενα FreqDistribution με μηδενική συχνότητα και μηδενικά έσοδα και αφορούν για το propertyFreqDistribution καθένα property, για το monthFreqDistribution κάθε μήνα του χρόνου, για το yearFreqDistribution όλα τα έτη μεταξύ του from και το που έχουν δοθεί σαν παράμετροι, για το weekdayFreqDistribution όλες τις μέρες τις εβδομάδας και για το genderFreqDistribution άντρες και γυναίκες.

```
// fill properties' frequency distribution list with all properties
for (Property p : properties) {
    propertyFreqDistribution.add(FreqDistribution.builder()
            .name(p.getPropertyName())
            .reservations(0)
            .income(0)
            .build());
}

Locale locale = new Locale( language: "en",  country: "US");

// fill years' frequency distribution list with all gears between from-to (including them)
for (int y = from; y <= to; y++) {
    yearFreqDistribution.add(FreqDistribution.builder()
            .name(String.valueOf(y))
            .reservations(0)
            .income(0)
            .build());
}


// fill months' frequency distribution list with all months
List<String> shortMonths = Arrays.asList(new DateFormatSymbols(locale).getShortMonths()).subList(0, 12);
for (String monthFormat : shortMonths) {
    monthFreqDistribution.add(FreqDistribution.builder()
            .name(monthFormat)
            .reservations(0)
            .income(0)
            .build());
}
```

78. Αρχικοποίηση των στατιστικών μέτρων στη getStats μέθοδο του Reservation Service (1)

```
// fill weekdays' frequency distribution list with all weekdays
List<String> weekdays = new ArrayList<>(Arrays.asList(new DateFormatSymbols(locale).getShortWeekdays()).subList(1, 8));
// The list we get has  Sunday as the first day, so we move it at the end
String sunday = weekdays.remove( index: 0);
weekdays.add(sunday);
for (String day : weekdays) {
    System.out.println(weekdays);
    weekdayFreqDistribution.add(FreqDistribution.builder()
            .name(day)
            .reservations(0)
            .income(0)
            .build());
}

// fill genders' frequency distribution list with the 2 genders
List<String> genders = Arrays.asList("Male", "Female");
for (String gender : genders) {
    genderFreqDistribution.add(FreqDistribution.builder()
            .name(gender)
            .reservations(0)
            .income(0)
            .build());
}
```

79. Αρχικοποίηση των στατιστικών μέτρων στη getStats μέθοδο του Reservation Service (2)

Στη συνέχεια, ανατρέχουμε την λίστα με τις κρατήσεις που έχουμε σαν παράμετρο. Η διαδικασία για όλες τις λίστες των FreqDistribution αντικειμένων είναι παρόμοια, επομένως θα περιγραφεί η propertyFreqDistribution καθώς η επεξήγηση των υπολοίπων μπορεί να εννοηθεί.

Πρώτα προσθέτουμε για κάθε κράτηση τα έσοδα της για τον υπολογισμό του συνόλου. Ύστερα, από την propertyFreqDistribution λίστα, κρατάμε το FreqDistribution αντικείμενο που αντιστοιχεί στο property της τρέχουσας κράτησης. Εάν βρεθεί, καλούμε την setFrequency μέθοδο η οποία δέχεται το FreqDistribution που βρήκαμε και την κράτηση.

```
for (Reservation r : reservations) {
    totalIncome += r.getTotalPrice();

    //Properties Statistics

    // fetch the property statistics record of the property of the reservation
    List<FreqDistribution> existingPropertyFrequency = propertyFreqDistribution.stream().filter(ps -> ps.getName().equals(r.getProperty().getPropertyName()))
            .toList();

    if (!existingPropertyFrequency.isEmpty()) {
        setFrequency(existingPropertyFrequency.get(0), r);
    }
```

80. Συμπλήρωση στατιστικών στη getStats μέθοδο του Reservation Service

Στην **setFrequency** μέθοδο, αυξάνεται ο αριθμός των κρατήσεων που αντιστοιχούν στο property κατά ένα και προσθέτουμε στα συνολικά του property τα έσοδα της κράτησης.

```
private void setFrequency(FreqDistribution freqDistribution, Reservation reservation) {
    freqDistribution.setReservations(freqDistribution.getReservations() + 1);
    freqDistribution.setIncome(freqDistribution.getIncome() + reservation.getTotalPrice());
}
```

81. setFrequency Μέθοδος του Reservation Service

Στο τέλος της getStats, συναρμολογούμε το ReservationStatistics αντικείμενο περνώντας τις λίστες, την τιμή των συνολικών εσόδων και το σύνολο των κρατήσεων στα αντίστοιχα πεδία.

```
return ReservationsStatistics.builder()
        .totalReservations(reservations.size())
        .totalIncome(totalIncome)
        .propertyFreqDistribution(propertyFreqDistribution)
        .monthFreqDistribution(monthFreqDistribution)
        .yearFreqDistribution(yearFreqDistribution)
        .weekdayFreqDistribution(weekdayFreqDistribution)
        .genderFreqDistribution(genderFreqDistribution)
        .build();
```

82. Επιστροφή στατιστικών στη getStats μέθοδο του Reservation Service

2.7.6. Reservation Controller

Στον **ReservationController**, κάνουμε Autowire τα απαραίτητα services. Αυτά είναι το ReservationService, το ClientService, το PropertyService, το RoomService, το TableService, το CarService και το EmailSenderService.

```java
@CrossOrigin
@RestController
@RequestMapping("/vaccoon-api/reservation")
public class ReservationController {
    14 usages
    @Autowired
    private ReservationService reservationService;
    3 usages
    @Autowired
    private ClientService clientService;
    2 usages
    @Autowired
    private PropertyService propertyService;
    1 usage
    @Autowired
    private RoomService roomService;
    1 usage
    @Autowired
    private TableService tableService;
    1 usage
    @Autowired
    private CarService carService;

    1 usage
    @Autowired
    EmailSenderService emailSenderService;
```

83. Reservation Controller

Η **getReservations** μέθοδος, εκτελείται όταν η εφαρμογή δέχεται **GET** requests στο *"vaccoon-api/reservation/"* endpoint. Δέχεται σαν Request Parameters τις τιμές from και to. Οι παράμετροι αυτοί δεν είναι υποχρεωτικό να υπάρχουν. Παίρνει τον συνδεδεμένο χρήστη από το SecurityContextHolder και καλεί την getReservations μέθοδο του ReservationService η οποία επιστρέφει μία λίστα κρατήσεων. Επιστρέφεται ένα Response Entity με response code **OK** και την λίστα από τις κρατήσεις στο body.

```
@GetMapping({"/", ""})
public ResponseEntity<List<Reservation>> getReservations(@RequestParam(value = "from", required = false) LocalDate fromDate,
                                                         @RequestParam(value = "to", required = false) LocalDate toDate) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    User signedUser = (User) authentication.getPrincipal();
    List<Reservation> reservations = reservationService.getReservations(fromDate, toDate, signedUser);
    return ResponseEntity.ok(reservations);
}
```

84. getReservations μέθοδος του Reservation Controller

Η μέθοδος **getReservation**, εκτελείται όταν δέχεται η εφαρμογή **GET** request στο endpoint *"vaccoon-api/reservation/{reservation-id}"* , όπου reservation id είναι το αναγνωριστικό της κράτησης που θα αναζητήσουμε. Παίρνουμε πρώτα των συνδεδεμένο χρήστη από το SecurityContextHolder και καλούμε την getReservation μέθοδο του ReservationService. Εάν η κράτηση βρεθεί, επιστρέφουμε ένα Response Entity με response code **OK** και στο body την κράτηση που βρέθηκε. Εάν δεν βρεθεί, επιστρέφει ένα Response Entity με κωδικό **No Content**.

```
@GetMapping("/{reservation_id}")
public ResponseEntity<Reservation> getReservation(@PathVariable("reservation_id") int reservation_id) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    User signedUser = (User) authentication.getPrincipal();
    Reservation reservation;
    try {
        reservation = reservationService.getReservation(reservation_id, signedUser);
    } catch (ResevationNotFoundException e) {
        return ResponseEntity.noContent().build();
    }
    return ResponseEntity.ok(reservation);
}
```

85. getReservation μέθοδος του Reservation Controller

Η **getStatistics** μέθοδος, εκτελείται όταν η εφαρμογή δέχεται **GET** requests στο *"vaccoon-api/reservation/stats"* endpoint. Δέχεται σαν παραμέτρους αρχικό και τελικό έτος, τύπο περιόδου αναζήτησης και τον τύπο του property. Ο τύπος περιόδου αφορά ετήσιο ή μηνιαίο. Ο μηνιαίος τύπος δεν απαιτεί τελικό έτος καθώς η περίοδος είναι το αρχικό έτος. Παίρνει πρώτα τον χρήστη από το SecurityContextHolder, και ορίζει μέγιστο και ελάχιστο περιορισμό έτους. Εάν το αρχικό έτος είναι null ή εκτός των περιορισμών, το θέτουμε ίσο με το τρέχον έτος. Στη συνέχεια, εάν το τελικό έτος είναι null, εκτός περιορισμών ή ο τύπος περιόδου είναι μηνιαίος, το τελικό έτος παίρνει τιμή ίση με το αρχικό.

Αφού έχει υπολογισθεί η περίοδος, παίρνουμε τις κρατήσεις αυτής καλώντας την μέθοδο getReservationsFromYearPeriod του ReservationService. Στη συνέχεια συλλέγουμε όλα τα properties του τύπου που έχει ορισθεί. Τέλος καλούμε την getStats του ReservationService το αποτέλεσμα της οποίας περνάμε στο body του response που επιστρέφουμε με response code **OK**.

```
@GetMapping("/stats")
public ResponseEntity<ReservationsStatistics> getStatistics(@RequestParam(value = "from", required = false) Integer fromYear,
                                                             @RequestParam(value = "to", required = false) Integer toYear,
                                                             @RequestParam(value = "period", defaultValue = "monthly") String period,
                                                             @RequestParam(value = "propertyType", defaultValue = "room") String propertyType)
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    User signedUser = (User) authentication.getPrincipal();

    int THRESHOLD_YEAR_MIN = 1950;
    int THRESHOLD_YEAR_MAX = 2090;

    if (fromYear == null || fromYear < THRESHOLD_YEAR_MIN || fromYear > THRESHOLD_YEAR_MAX) {
        fromYear = Year.now().getValue();
    }

    if (period.equals("monthly") || toYear == null || toYear < THRESHOLD_YEAR_MIN || toYear > THRESHOLD_YEAR_MAX || fromYear > toYear) {
        toYear = fromYear;
    }

    List<Reservation> reservations = reservationService.getReservationsFromYearPeriod(fromYear, toYear, propertyType, signedUser);
    List<Property> properties = new ArrayList<>();

    switch (propertyType) {
        case "room" -> properties.addAll(roomService.getAllRooms( fromDate: null,  toDate: null, signedUser));
        case "table" -> properties.addAll(tableService.getAllTables( fromDate: null, signedUser));
        case "car" -> properties.addAll(carService.getAllCars( fromDate: null,  toDate: null, signedUser));
    }

    return ResponseEntity.ok(reservationService.getStats(reservations, properties, fromYear, toYear));
}
```

86. getStatistics μέθοδος του Reservation Controller

Η **addReservation** μέθοδος, εκτελείται όταν η εφαρμογή δέχεται **POST** requests στο *"vaccoon-api/reservation"* endpoint. Απαραίτητο στοιχείο στο request είναι η παρουσία ενός JSON στη μορφή του ReservationReq. Πρώτα παίρνουμε τον χρήστη

από το SecurityContextHolder. Στη συνέχεια παίρνουμε το ClientReq αντικείμενο από το ReservationReq. Το αντικείμενο αυτό περνάει σταν παράμετρος στην private μέθοδο του Controller, getClient, η οποία ελέγχει αν υπάρχει πελάτης με τα στοιχεία που υπάρχουν στο ClientReq και τον επιστρέφει. Αν δεν υπάρχει, δημιουργεί έναν καινούργιο, τον αποθηκεύει και τον επιστρέφει.

```
@PostMapping
public ResponseEntity<Object> addReservation(@RequestBody @NotNull ReservationReq reservationReq) throws ClientEmailPhoneIncompatibleException {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    User signedUser = (User) authentication.getPrincipal();
    ClientReq clientReq = reservationReq.getClientreq();

    Client client = getClient(clientReq, signedUser);
```

87. addReservation μέθοδος του Reservation Controller (1)

Στη συνέχεια, ελέγχουμε ότι το property id του ReservationReq αντιστοιχεί σε υπάρχον property, υπολογίζουμε τα συνολικά έσοδα, δημιουργούμε και αποθηκεύουμε την κράτηση και στέλνουμε ένα mail στον χρήστη ,έσω του sendEmail του EmailSenderService.

```
Property property;
try {
    property = propertyService.getProperty(reservationReq.getPropertyId(), signedUser);
} catch (PropertyNotFoundException e) {
    return ResponseEntity.noContent().build();
}

float totalPrice;
if (reservationReq.getToDate() != null && reservationReq.getTotalPrice() == 0) {
    long daysDuration = ChronoUnit.DAYS.between(reservationReq.getFromDate(), reservationReq.getToDate());
    totalPrice = property.getPrice() * daysDuration;
} else {
    totalPrice = reservationReq.getTotalPrice();
}

Reservation reservation = Reservation.builder()
        .reservationDate(LocalDateTime.now())
        .property(property)
        .client(client)
        .user(signedUser)
        .status(ReservationStatus.PENDING)
        .numOfPeople(reservationReq.getNumOfPeople())
        .fromDate(reservationReq.getFromDate())
        .toDate(reservationReq.getToDate())
        .checkIn(null)
        .checkOut(null)
        .totalPrice(totalPrice)
        .propertyType(reservationReq.getPropertyType())
        .build();

reservationService.saveReservation(reservation);

emailSenderService.sendEmail(reservation.getClient().getEmail(), subject: "New reservation", buildReservationEmail(reservation));

return ResponseEntity.ok().build();
```

88. addReservation μέθοδος του Reservation Controller (2)

Το περιεχόμενο του email το βρίσκουμε στη μέθοδο buildReservationEmail:

```
private String buildReservationEmail(Reservation reservation) {
    return """
            <div>
                <h2>Dear %s, </h2>
            </div
            <br>
            <div style="font-size: 1.2rem">
                <i>You have a new %s reservation.</i>
            </div>
            <div style="background-color: rgba(B, B, B, 0.7); font-size: 0.8rem; margin: 10px 30px; padding: 10px 20px">
                <h4><u>Reservation details: </u></h4>
                <b>%s</b>: %s <br>
                <b>from:</b> %s <br>
                <b>to:</b> %s <br>
                <b>people:</b> %s
            </div>
            <br>
            <h3>Scan QR at check in</h3>
            <div style="text-align: center">
                <img id='barcode' src="https://api.qrserver.com/v1/create-qr-code/?data=%s&amp;size=100x100" alt="" width="250" height="250" />
            </div>
            """.formatted(
            reservation.getClient().getFirstname(),
            reservation.getPropertyType(),
            reservation.getPropertyType(),
            reservation.getProperty().getPropertyName(),
            reservation.getFromDate(),
            reservation.getToDate() == null ? "-" : reservation.getToDate(),
            reservation.getNumOfPeople(),
            reservation.getReservationId()
    );
}
```

89. buildReservationEmail μέθοδος του Reservation Controller

Η μέθοδος **updateReservation**, καλείται όταν η εφαρμογή δέχεται **PUT** requests στο endpoint *"vaccoon-api/reservation/{reservation-id}"*. Το **reservation-id** είναι ένα path variable το οποίο περνάει σαν παράμετρος στην μέθοδο. Επίσης η μέθοδος περιμένει ένα JSON αντικείμενο στη μορφή του ReservationReq. Πρώτα παίρνουμε τον χρήστη από το SecurityContextHolder. Ελέγχεται εάν υπάρχει χρήστης με τα στοιχεία που υπάρχουν στο ClientReq και εάν δεν υπάρχει, δημιουργείται και αποθηκεύεται. Ύστερα, παίρνουμε το Property που αντιστοιχεί στο property id του ReservationReq. Εάν δεν βρεθεί εγγραφή επιστρέφεται Response Entity με response code **No Content**. Στην συνέχεια καλείται η getReservation μέθοδος της ReservationService, με παραμέτρους το reservation id και τον χρήστη και κρατάμε σε ένα αντικείμενο την εγγραφή που αντιστοιχούν. Εάν δεν βρεθεί κάποια τέτοια εγγραφή στη βάση, όπως αναφέραμε και στα services, παίρνουμε ένα ReservationNotFoundException το οποίο διαχειριζόμαστε επιστρέφοντας ένα ResponseEntity με response code **No Content** (204). Αλλιώς, ενημερώνουμε τις τιμές του αντικειμένου σύμφωνα με τις τιμές του ReservationReq που έχει δοθεί και επιστρέφουμε **OK** (200) με μία true Boolean μεταβλητή.

```java
@PutMapping("/{reservation_id}")
public ResponseEntity<Boolean> updateReservation(@RequestBody @NotNull ReservationReq reservationReq, @PathVariable("reservation_id") int reservation_id)
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    User signedUser = (User) authentication.getPrincipal();

    ClientReq clientReq = reservationReq.getClientreq();
    Client client = getClient(clientReq, signedUser);

    Property property;
    try {
        property = propertyService.getProperty(reservationReq.getPropertyId(), signedUser);
    } catch (PropertyNotFoundException e) {
        return ResponseEntity.noContent().build();
    }

    Reservation reservation;
    try {
        reservation = reservationService.getReservation(reservation_id, signedUser);
    } catch (ResevationNotFoundException e) {
        return ResponseEntity.noContent().build();
    }

    reservation.setProperty(property);
    reservation.setClient(client);
    reservation.setNumOfPeople(reservationReq.getNumOfPeople());
    reservation.setFromDate(reservationReq.getFromDate());
    reservation.setToDate(reservationReq.getToDate());
    reservation.setCheckIn(reservationReq.getCheckIn());
    reservation.setCheckOut(reservationReq.getCheckOut());
    reservation.setTotalPrice(reservationReq.getTotalPrice());
    reservation.setStatus(reservationReq.getStatus());

    reservationService.updateReservation(reservation);

    return ResponseEntity.ok( body: true);
}
```

90. updateReservation μέθοδος του Reservation Controller

Η μέθοδος **setReservationCheckIn**, καλείται όταν η εφαρμογή δέχεται **PUT** requests στο endpoint *"vaccoon-api/reservation/{reservation-id}/checkin"*. Πρώτα παίρνουμε τον χρήστη από το SecurityContextHolder και από το reservation id, βρίσκουμε την κράτηση που αντιστοιχεί σε αυτό. Αν δεν βρεθεί επιστρέφουμε response με response code **No Content**. Αν βρεθεί, ενημερώνουμε την κατάστασή της σε **CHECKED IN** και την ημερομηνία και ώρα το check in ίση με την τρέχουσα. Το ενημερωμένο αντικείμενο το δίνουμε σαν παράμετρο στην updateReservation μέθοδο του ReservationService η οποία θα ενημερώσει την εγγραφή της στη βάση.

```java
@PutMapping("/{reservation_id}/checkin")
public ResponseEntity<Boolean> setReservationCheckIn(@PathVariable("reservation_id") int reservation_id) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    User signedUser = (User) authentication.getPrincipal();

    Reservation reservation;
    try {
        reservation = reservationService.getReservation(reservation_id, signedUser);
    } catch (ResevationNotFoundException e) {
        return ResponseEntity.noContent().build();
    }

    reservation.setStatus(ReservationStatus.CHECKED_IN);
    reservation.setCheckIn(LocalDateTime.now());

    reservationService.updateReservation(reservation);

    return ResponseEntity.ok( body: true);
}
```

91. setReservationCheckIn μέθοδος του Reservation Controller

Η μέθοδος **setReservationCheckOut**, καλείται όταν η εφαρμογή δέχεται **PUT** requests στο endpoint *"vaccoon-api/reservation/{reservation-id}/checkout"*. Πρώτα παίρνουμε τον χρήστη από το SecurityContextHolder και από το reservation id, βρίσκουμε την κράτηση που αντιστοιχεί σε αυτό. Αν δεν βρεθεί επιστρέφουμε response με response code **No Content**. Αν βρεθεί, ενημερώνουμε την κατάστασή της σε **CHECKED OUT** και την ημερομηνία και ώρα το check out ίση με την τρέχουσα. Το ενημερωμένο αντικείμενο το δίνουμε σαν παράμετρο στην updateReservation μέθοδο του ReservationService η οποία θα ενημερώσει την εγγραφή της στη βάση.

```java
@PutMapping("/{reservation_id}/checkout")
public ResponseEntity<Boolean> setReservationCheckOut(@PathVariable("reservation_id") int reservation_id) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    User signedUser = (User) authentication.getPrincipal();

    Reservation reservation;

    try {
        reservation = reservationService.getReservation(reservation_id, signedUser);
    } catch (ResevationNotFoundException e) {
        return ResponseEntity.noContent().build();
    }

    if (!ReservationStatus.CHECKED_IN.equals(reservation.getStatus())) {
        return ResponseEntity.ok( body: false);
    }

    reservation.setStatus(ReservationStatus.CHECKED_OUT);
    reservation.setCheckOut(LocalDateTime.now());

    reservationService.updateReservation(reservation);
    return ResponseEntity.ok( body: true);
}
```

92. setReservationCheckOut μέθοδος του Reservation Controller

Η μέθοδος **cancelReservation**, καλείται όταν η εφαρμογή δέχεται **PUT** requests στο endpoint *"vaccoon-api/reservation/{reservation-id}/cancel"*. Πρώτα παίρνουμε τον χρήστη από το SecurityContextHolder και από το reservation id, βρίσκουμε την κράτηση που αντιστοιχεί σε αυτό. Αν δεν βρεθεί επιστρέφουμε response με response code **No Content**. Αν βρεθεί, ενημερώνουμε την κατάστασή της σε **CANCELED**. Το ενημερωμένο αντικείμενο το δίνουμε σαν παράμετρο στην updateReservation μέθοδο του ReservationService η οποία θα ενημερώσει την εγγραφή της στη βάση.

```
@PutMapping("/{reservation_id}/cancel")
public ResponseEntity<Boolean> cancelReservation(@PathVariable("reservation_id") int reservation_id) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    User signedUser = (User) authentication.getPrincipal();

    Reservation reservation;

    try {
        reservation = reservationService.getReservation(reservation_id, signedUser);
    } catch (ResevationNotFoundException e) {
        return ResponseEntity.noContent().build();
    }

    reservation.setStatus(ReservationStatus.CANCELED);

    reservationService.updateReservation(reservation);
    return ResponseEntity.ok( body: true);
}
```

93. cancelReservation μέθοδος του Reservation Controller

Η μέθοδος **deleteReservation**, καλείται όταν η εφαρμογή δέχεται **DELETE** requests στο endpoint *"vaccoon-api/reservation/{reservation-id}"*. Πρώτα παίρνουμε τον χρήστη από το SecurityContextHolder. Δημιουργούμε ένα νέο αντικείμενο Reservation με το reservation id και τον χρήστη και το περνάμε σαν παράμετρο στη μέθοδο deleteReservation του ReservationService. Επιστρέφουμε ένα Response Entity με response code **OK** και στο body μία μεταβλητή Boolean η τιμή της οποίας εξαρτάται απ' το εάν η τιμή που επέστρεψε η μέθοδος του service ήταν μεγαλύτερη από 0.

```
@DeleteMapping("/{reservation_id}")
public ResponseEntity<Boolean> deleteReservation(@PathVariable("reservation_id") int reservation_id) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    User signedUser = (User) authentication.getPrincipal();

    Reservation reservation = Reservation.builder()
            .reservationId(reservation_id)
            .user(signedUser)
            .build();

    return ResponseEntity.ok( body: reservationService.deleteReservation(reservation) > 0);
}
```

94. deleteReservation μέθοδος του Reservation Controller

2.8. Email

2.8.1. Email Service

Η **EmailSenderService** κλάση έχει την **sendEmail** μέθοδο, υπεύθυνη για την αποστολή email με παραλήπτη, θέμα και κείμενο που θα δεχθεί σαν παραμέτρους. Συγκεκριμένα, για την υλοποίηση χρησιμοποιούμε την **JavaMailSender** και το **MimeMessage** αντικείμενο που μας παρέχει. Με το MimeMessage μπορούμε να περάσουμε στο σώμα του email μας HTML κώδικα, κάνοντάς το έτσι πιο περιεκτικό και αισθητικό. Περνάμε στο MimeMessage τα στοιχεία που πήραμε από τις παραμέτρους της συνάρτησης και με την μέθοδο **send** του JavaMailSender, στέλνουμε το mail.

```
@Service
public class EmailSenderService {
    1 usage
    private final static Logger log = LoggerFactory.getLogger(EmailSenderService.class);
    2 usages
    @Autowired
    private JavaMailSender javaMailSender;

    2 usages  chrisleskos
    public void sendEmail(String toAddress, String subject, String body) {
        try {
            MimeMessage mimeMessage = javaMailSender.createMimeMessage();
            MimeMessageHelper mimeHelper = new MimeMessageHelper(mimeMessage, encoding: "utf-8");
            mimeHelper.setTo(toAddress);
            mimeHelper.setSubject(subject);
            mimeHelper.setText(body, html: true);

            javaMailSender.send(mimeMessage);
        } catch (MessagingException e) {
            log.error("Failed to send email", e);
            throw new IllegalStateException("Failed to send email");
        }
    }
}
```

95. Email Sender Service

3. Τεχνική ανάλυση υλοποίησης διεπαφής χρήστη

3.1. Εισαγωγή

Για το γραφικό περιβάλλον και την διεπαφή του χρήστη με το σύστημα, χρησιμοποιείται η **ReactJS** μέσω **Vite JS**. Ο κώδικας των αρχείων έχει γραφτεί σε TypeScript, μία γλώσσα προγραμματισμού που επεκτείνει την JavaScript.

Μία από τις σημαντικές ιδιαιτερότητες της React είναι ότι μπορούμε να γράψουμε αρχεία **Components**, τα οποία είναι κώδικας για την δομή, λειτουργία και rendering μίας σελίδας ή ενός στοιχείου (παράδειγμα ένα Alert Box), ο οποίος μπορεί να ξαναχρησιμοποιηθεί σε άλλο αρχείο/σελίδα καθώς και να παραμετροποιηθεί σε περίπτωση απαραίτητης προσαρμογής.

Οι σελίδες επικοινωνούν με το Vaccoon API μέσω του **Axios**, μίας βιβλιοθήκης της JavaScript όπου επιτρέπει την αποστολή HTTP request σε κάποιον εξυπηρετητή. Η διαδικασία αυτή γίνεται **ασύγχρονά**. Αυτό σημαίνει πως αφού γίνει η κλήση, η βασική ροή του προγράμματος δεν περιμένει την ολοκλήρωσή της, και όταν αυτή επέλθει, τότε θα εκτελεστεί ο κώδικας που έχει γραφτεί για αυτή την περίπτωση.

Η React, χρησιμοποιεί επίσης διάφορα **Hooks**. Τα Hooks είναι συναρτήσεις οι οποίες έχουν πρόσβαση στο State object της React. Το State περιέχει πληροφορίες για την κατάσταση της σελίδας και των components. Με αυτόν τον τρόπο, μπορούμε να ορίσουμε και να προσθέσουμε μεταβλητές και μεθόδους στο State object. Η ιδιαιτερότητα έγκειται στο ότι, όταν αλλάξει κάποια πληροφορία στο State, η React σελίδα γίνεται **re-render**, έχοντας έτσι «ζωντανή» ενημέρωση της κατάστασης των μεταβλητών.

Για την είσοδο στις σελίδες της εφαρμογής, χρειάζεται ο χρήστης να είναι συνδεδεμένος. Αυτό ελέγχεται μέσω των cookies. Εάν έχει γίνει επιτυχής σύνδεση στο API, αυτό απαντάει με ένα JWT. Το token αυτό αποθηκεύεται στα cookies του browser του χρήστη. Σε κάθε προσπάθεια επίσκεψης μίας σελίδας, γίνεται έλεγχος ότι το cookie είναι παρών και πως η τιμή του είναι έγκυρη.

3.2. Βασική δομή React

3.2.1. Environment Properties

Το property file είναι ένα αρχείο στο οποίο βρίσκονται τιμές που μπορεί να αλλάξουν ανά περιβάλλον εκτέλεσης της εφαρμογής ή από άλλους παράγοντες. Γενικά οι τιμές που περιέχονται δεν θέλουμε να είναι hard coded στο πρόγραμμα μας. Έτσι, βρίσκουμε το αρχείο *".env"* στο οποίο έχουμε γράψει τις μεταβλητές περιβάλλοντος.

```
VITE_API_HOST=http://192.168.1.14
VITE_API_PORT=8081
VITE_API_PATH=/vaccoon-api
```

96. Environment Properties

Οι μεταβλητές αυτές είναι το host στο οποίο ζει το API, το Port του, καθώς και το κεντρικό path της εφαρμογής.

3.2.2. App.tsx

Το **App.tsx** είναι το πρώτο αρχείο που διαβάζεται όταν φορτώνουμε την σελίδα μας.

```
import HandleRoutes from "./HandleRoutes";

function App() {
  const host = import.meta.env.VITE_API_HOST;
  const port = import.meta.env.VITE_API_PORT;
  const vaccoonApiPath = import.meta.env.VITE_API_PATH;
  const baseUrl = host + ":" + port + vaccoonApiPath;
  return (
    <div className="App">
      <HandleRoutes baseUrl={baseUrl} />
    </div>
  );
}

export default App;
```

97. App.tsx

Αυτό που κάνει είναι να φορτώνει από το αρχείο με τις μεταβλητές περιβάλλοντος το host, port και path και να τα ενώνει ώστε να έχουμε το βασικό URL για τις κλήσεις στο API. Ύστερα, φορτώνει το **HandleRoutes** component

3.2.3. Εναλλαγή σελίδων/Routing

Στο **HandleRoutes** component, διαχειριζόμαστε την ανακατεύθυνση του χρήστη στις σελίδες τις εφαρμογής ανάλογα με το αν είναι συνδεδεμένος. Αυτό γίνεται διαβάζοντας το cookie με όνομα "token" στο οποίο έχουμε μέσα την τιμή του JWT. Με το useEffect hook, μπορούμε να τρέξουμε κώδικα πριν γίνει render η σελίδα. Έτσι τρέχουμε πρώτα ένα Axios call στο *"vaccoon-api/auth/check"* path του API για να ελέγξουμε εάν το token είναι έγκυρο.

```
function HandleRoutes({ baseUrl }: HandleRoutesProps) {
  const checkIfLoggedUrl = "/auth/check";
  const [cookies] = useCookies(["token"]);
  const token = cookies.token;

  const [isTokenValid, setTokenValid] = useState(token !== undefined);
  useEffect(() => {
    Axios.get(baseUrl + checkIfLoggedUrl, {
      headers: {
        Authorization: "Bearer " + token,
      },
    })
      .then(function (response) {
        setTokenValid(response.data);
      })
      .catch(function (error) {
        setTokenValid(false);
      });
  }, [baseUrl, checkIfLoggedUrl]);
```

98. HandleRoutes.tsx

Την εγκυρότητα του token την φορτώνουμε στην **isTokenValid**, μία μεταβλητή useState Hook. Αν επιστραφεί response χωρίς πρόβλημα, τότε είναι έγκυρο, αλλιώς δεν είναι.

Ανάλογα την τελευταία συνθήκη, έχουμε και τον διαχωρισμό της δρομολόγησης. Συγκεκριμένα, εάν δεν είναι έγκυρο, οποιαδήποτε κλήση στην εφαρμογή, εκτός της *"/register"* που μας εμφανίζει την σελίδα εγγραφής, μας πηγαίνει στην σελίδα σύνδεσης. Εάν το token είναι έγκυρο, έχουμε την κανονική δρομολόγηση του χρήστη στις σελίδες.

```
if (!isTokenValid) {
  return (
    <Routes>
      <Route
        path="/register"
        element={<RegisterPage baseUrl={baseUrl} />}
      ></Route>
      <Route path="*" element={<LoginPage baseUrl={baseUrl} />}></Route>
    </Routes>
  );
} else {
  return (
    <Routes>
      <Route path="/" element={<HomePage baseUrl={baseUrl} />} />
      <Route path="/login" element={<HomePage baseUrl={baseUrl} />} />
      <Route path="/register" element={<HomePage baseUrl={baseUrl} />} />
      <Route path="/home" element={<HomePage baseUrl={baseUrl} />} />
      <Route path="/rooms" element="">
        <Route index element={<RoomsPage baseUrl={baseUrl} />} />
        <Route path="new" element={<NewRoomPage baseUrl={baseUrl} />} />
        <Route
          path=":propertyId"
          element={<GetRoomPage baseUrl={baseUrl} />}
        ></Route>
      </Route>
      <Route path="/restaurant" element="">
        <Route index element={<RestaurantPage baseUrl={baseUrl} />} />
        <Route path="new" element={<NewTablePage baseUrl={baseUrl} />} />
        <Route
          path=":propertyId"
          element={<GetTablePage baseUrl={baseUrl} />}
        />
      </Route>
      <Route path="/cars" element="">
```

99. Δρομολόγηση αιτημάτων στη React

3.3. Παράδειγμα υλοποίησης σελίδας

Για την περιγραφή της υλοποίησης ως παράδειγμα θα χρησιμοποιηθεί το αρχείο **GetClientPage.tsx**.

Στην αρχή του κώδικα, ορίζουμε ένα useParams Hook, το **urlParam**, με το οποίο θα μορούμε να διαβάσουμε τυχόν παραμέτρους στο URL της κλήσης της σελίδας. Στη συνέχεια ορίζουμε το path για τις κλήσεις του API που αφορούν την σελίδα και φορτώνουμε το "token" cookie. Παρακάτω βλέπουμε πως ορίζουμε κάποια useState Hooks τα οποία θα κρατάνε τιμές για την κατάσταση του Alert Box component που θα δούμε παρακάτω. Επίσης, ορίζουμε το **navigate** ως useNavigate Hook, το οποίο θα μας βοηθήσει να ανακατευθυνθούμε σε άλλη σελίδα όταν αυτό θα είναι απαραίτητο.

```
function GetClientPage({ baseUrl }: GetClientPageProps) {
  const urlParam = useParams();
  const clientUrl = "/client";
  const [cookies] = useCookies(["token"]);
  const token = cookies.token;

  const navigate = useNavigate();

  const [showAlert, setShowAlert] = useState(false);
  const [alertMessage, setAlertMessage] = useState("");
  const [alertType, setAlertType] = useState("");

  const handleClick = () => {
    setShowAlert(false);
  };

  const [editMode, setEditMode] = useState(false);

  const [client, setClient] = useState<Client>({
    clientId: -1,
    firstname: "",
    lastname: "",
    email: "",
    phoneNum: "",
    gender: "",
    yearOfBirth: 0,
    country: "",
    comments: "",
    impression: 0,
  });
```

100. GetClientPage

Η **handleClick** μέθοδος, θα περαστεί σαν παράμετρος στο Alert Box component. Αυτό μας επιτρέπει να ορίζουμε το πως θα δράσει το πάτημα ενός κουμπιού ή κάποια άλλη αλληλεπίδραση με το component, από το Parent αρχείο που το καλεί.

Στη συνέχεια, ορίζουμε το **editMode** state hook, το οποίο θα μας χρησιμεύσει στο να εμφανίζουμε την φόρμα επεξεργασίας των στοιχείων του πελάτη.

Το **client** είναι ένα useState Hook, το οποίο κρατάει τα στοιχεία του πελάτη. Οι αρχικές τιμές του είναι όλες κενές.

Η μέθοδος **submitEditClient** είναι υπεύθυνη για την υποβολή της φόρμας επεξεργασίας του πελάτη. Δέχεται σαν παραμέτρους τα πεδία της φόρμας και, αφού πάρει τις τιμές που έχουν μεσα, εκτελεί ένα **PUT** request με την βιβλιοθήκη Axios [11]. Η κλήση γίνεται στο *"/client/{client-id}"* path του API, όπου σαν client id έχουμε δώσει το url parameter **clientId**. Στο header του request περνάμε το Authorization bearer με το token που έχουμε στα cookies.

```
// Take as parameters the input elements of the form
const submitEditClient = (
  e: React.FormEvent<HTMLFormElement>,
  firstname: React.RefObject<HTMLInputElement>,
  lastname: React.RefObject<HTMLInputElement>,
  email: React.RefObject<HTMLInputElement>,
  phoneNum: React.RefObject<HTMLInputElement>,
  gender: React.RefObject<HTMLSelectElement>,
  yearOfBirth: React.RefObject<HTMLSelectElement>,
  country: React.RefObject<HTMLSelectElement>,
  comments: React.RefObject<HTMLTextAreaElement>,
  impression: React.RefObject<HTMLInputElement>
) => {
  e.preventDefault();
  const firstnameVal = firstname.current!.value;
  const lastnameVal = lastname.current!.value;
  const emailVal = email.current!.value;
  const phoneNumVal = phoneNum.current!.value;
  const genderVal = gender.current!.value;
  const yearOfBirthVal = yearOfBirth.current!.value;
  const countryVal = country.current!.value;
  const commentsVal = comments.current!.value;
  const impressionVal = impression.current!.value;

  Axios.put(
    baseUrl + clientUrl + "/" + urlParam.clientId,
    {
      firstname: firstnameVal,
      lastname: lastnameVal,
      email: emailVal,
      phoneNum: phoneNumVal,
      gender: genderVal,
      yearOfBirth: yearOfBirthVal,
      country: countryVal,
      comments: commentsVal,
      impression: impressionVal,
    },
    {
      headers: {
```

101. submitEditClient μέθοδος του GetClientPage (1)

```
          Authorization: "Bearer " + token,
        },
      }
    )
      .then(function (response) {
        setAlertType("success");
        setAlertMessage("Client Updated Successfully");
        setShowAlert(true);
        firstname.current!.value = "";
        lastname.current!.value = "";
        email.current!.value = "";
        phoneNum.current!.value = "";
        gender.current!.value = "";
        yearOfBirth.current!.value = "";
        country.current!.value = "";
        comments.current!.value = "";
        impression.current!.value = "";
        navigate("/Clients");
      })
      .catch(function (error) {
        setAlertType("danger");
        setAlertMessage("Client email/phone already exists");
        setShowAlert(true);
      });
};
```

102. submitEditClient μέθοδος του GetClientPage (2)

Εάν η κλήση στο API είναι επιτυχής, εμφανίζουμε επιτυχές μήνυμα μέσω του Alert Box και αδειάζουμε τις τιμές των πεδίων. Εάν υπάρξει πρόβλημα, σημαίνει ότι ο χρήστης με αυτό το email ή τηλέφωνο υπάρχει ήδη και εμφανίζουμε ανεπιτυχές μήνυμα στο Alert Box.

Η **deleteClient** μέθοδος, στέλνει ένα **DELETE** request call με Axios στο endpoint διαγραφής του χρήστη. Κλείνει πρώτα το παράθυρο επιβεβαίωσης, το οποίο είχε ανοίξει και πατήθηκε η επιβεβαίωση διαγραφής για να προχωρήσει η κλήση της μεθόδου, και κάνει reload την σελίδα.

```
const deleteClient = () => {
  setShowModal(false);
  Axios.delete(baseUrl + clientUrl + "/" + client.clientId, {
    headers: {
      Authorization: "Bearer " + token,
    },
  }).then(function (response) {
    navigate("/Clients");
  });
};
```

103. deleteClient μέθοδος του GetClientPage

Το useEffect hook, καλεί την **loadClient** μέθοδο.Η loadClient διαβάζει το url parameter με το id του πελάτη και κάνει ένα **GET** request στο endpoint του API όπου επιστρέφει έναν πελάτη με αυτό το id. Αν βρεθεί, περνάμε τα στοιχεία του στη μεταβλητή client του useState hook.

```
useEffect(() => {
  loadClient(baseUrl, token, clientUrl, urlParam.clientId);
}, [baseUrl, token, clientUrl, urlParam.clientId]);

const loadClient = (
  baseUrl: string,
  token: string,
  ClientUrl: string,
  urlParam: string | undefined
) => {
  Axios.get(baseUrl + ClientUrl + "/" + urlParam, {
    headers: {
      Authorization: "Bearer " + token,
    },
  })
    .then(function (response) {
      if (response.data == "") {
        navigate("/error");
      }
      setClient(response.data);
    })
    .catch(function (error) {
      // removeCookies("token");
      // window.location.href = "/";
    });
};
```

104. Παράδειγμα useEffect Hook

Αφού ολοκληρωθεί η δήλωση μεταβλητών και μεθόδων, φτάνουμε στην δόμηση της σελίδας.

Πρώτα συμπεριλαμβάνουμε το **ConfirmDeleteModal** component το οποίο θα εμφανιστεί για την επιβεβαιωση διαγραφής του χρήστη. Στη συνέχεια έχουμε το **PageBase** component το οποίο περιέχει το navigation side bar και το header των σελίδων το οποίο είναι ίδιο σε όλες. Βλέπουμε ότι έχουμε προσθέσει HTML κώδικα μέσα στα PageBase tags. Ο κώδικας αυτός θα προστεθεί στο header της σελίδας, καθώς εκεί αναγράφεται σε ποια σελίδα βρίσκεται ο χρήστης, επομένως η πληροφορία αυτή θα πρέπει να παρέχεται ως παράμετρος.

```
return (
  <>
    <ConfirmDeleteModal
      showModal={showModal}
      onClose={handleCloseModal}
      onClick={deleteClient}
      title={modalTitle}
      body={modalBody}
      approveBtnText={modalApproveBtnText}
      approveBtnColor={modalApproveBtnColor}
    />
    <PageBase baseUrl={baseUrl}>
      <a href="/clients">Clients</a> {">"}{" "}
      <a href={"/clients/" + urlParam.clientId}>
        {client.firstname + " " + client.lastname}
      </a>{" "}
    </PageBase>
```

105. Επιστροφή σελίδας GetClientPage (1)

Στη συνέχεια, έχουμε το **MainContent** component στο οποίο συμπεριλαμβάνουμε το κεντρικό μέρος της σελίδας. Έχουμε αρχικά ένα Alert Box, με το όνομα του component να είναι **AlertMessage**. Παρακάτω, υπάρχει μία δήλωση με την χρήση της editMode μεταβλητής. Εάν είναι true, στην σελίδα εμφανίζεται η φόρμα επεξεργασίας. Από την άλλη, αν είναι false, εμφανίζεται η κάρτα με τα στοιχεία του πελάτη στην οποία δεν μπορούμε να τα επεξεργαστούμε.

```
<MainContent>
  <div className={styles["form-container"]}>
    {showAlert && (
      <AlertMessage
        onClose={handleClick}
        alertType={alertType}
        extraStyle={styles.alert}
      >
        {alertMessage}
      </AlertMessage>
    )}
    {editMode ? (
      <>
        <ClientForm
          baseUrl={baseUrl}
          onSubmit={submitEditClient}
          client={client}
        >
          Edit Client
        </ClientForm>
        <button
          className="btn btn-secondary"
          onClick={() => setEditMode(false)}
        >
          Cancel
        </button>
      </>
```

106. Επιστροφή σελίδας GetClientPage (2)

```
    ) : (
      <>
        <ClientDetails baseUrl={baseUrl} client={client}>
          <div>
            <button
              className="btn btn-primary"
              onClick={() => setEditMode(true)}
            >
              Edit
            </button>
            <button
              className="btn btn-danger"
              onClick={handleDeleteClick}
            >
              Delete
            </button>
          </div>
        </ClientDetails>
      </>
    )}
  </div>
</MainContent>
```

107. Επιστροφή σελίδας GetClientPage (3)

Τέλος, κάνουμε export το component **GetClientPage** για να μπορούμε να το χρησιμοποιήσουμε και σε άλλα components.

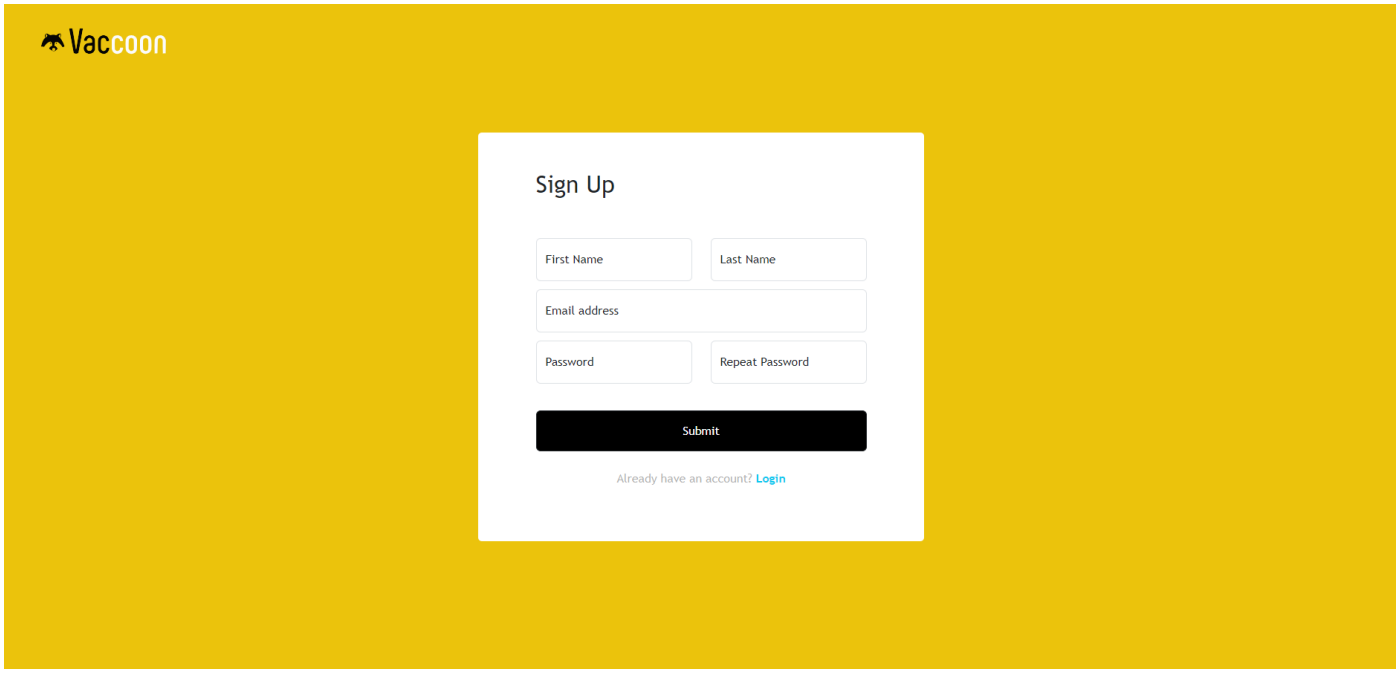
```
export default GetClientPage;
```

108. export GetClientPage

4. Περιγραφή λειτουργίας της προτεινόμενης εφαρμογής

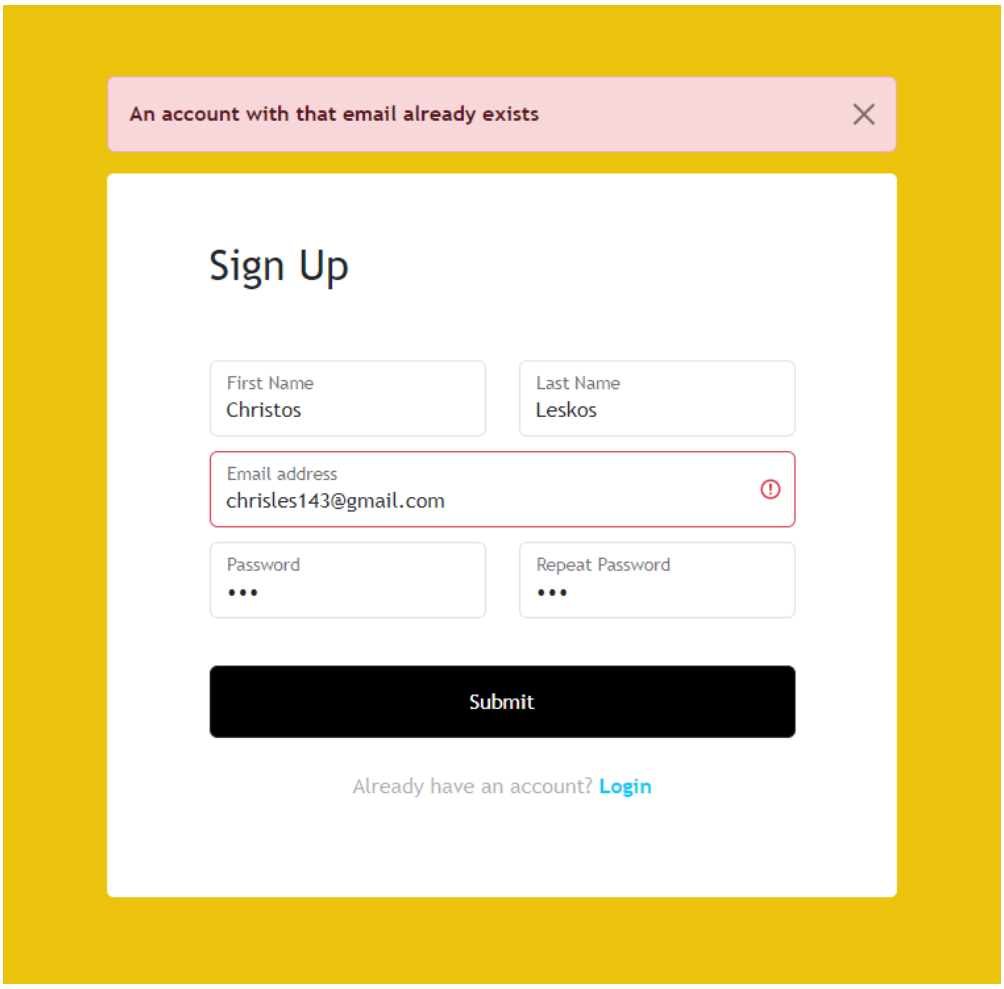
4.1. Εγγραφή

Στην σελίδα δημιουργίας νέου χρήστη, εισάγονται τα απαιτούμενα στοιχεία.



109. Σελίδα Εγγραφής

Εάν τα στοιχεία δεν περάσουν τους ελέγχους, όπως για παράδειγμα ότι υπάρχει ήδη χρήστη με αυτή την διεύθυνση email ή οι κωδικοί στα δύο αντίστοιχα πεδία δεν είναι ίδιοι, ο χρήστης ενημερώνεται με σχετικό μήνυμα.

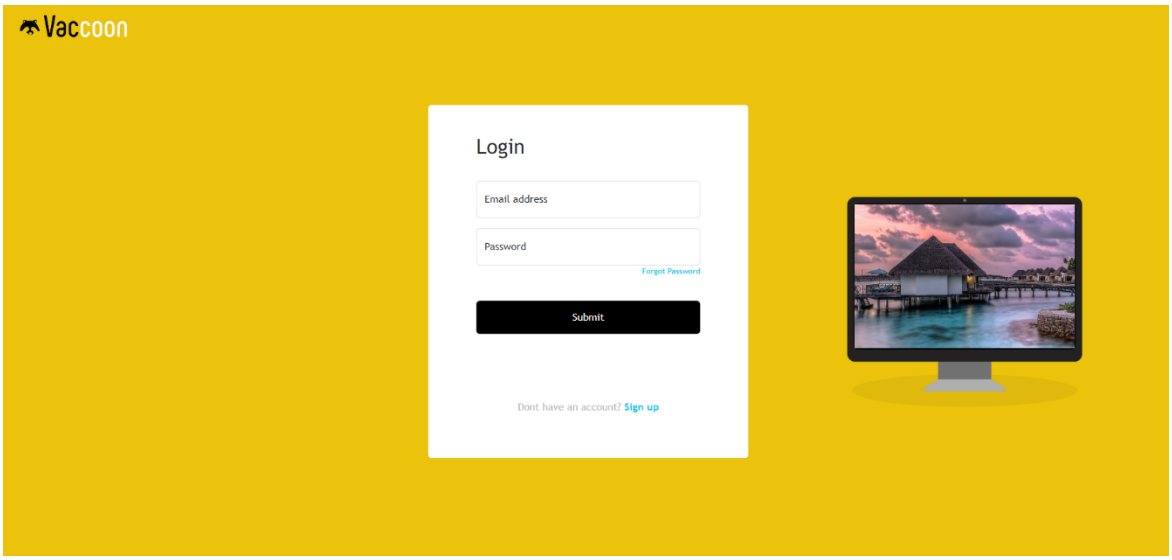


110. Παράδειγμα ανεπιτυχούς εγγραφής

Εάν οι έλεγχοι περάσουν επιτυχώς, ο χρήστης ανακατευθύνεται στην σελίδα Σύνδεσης.

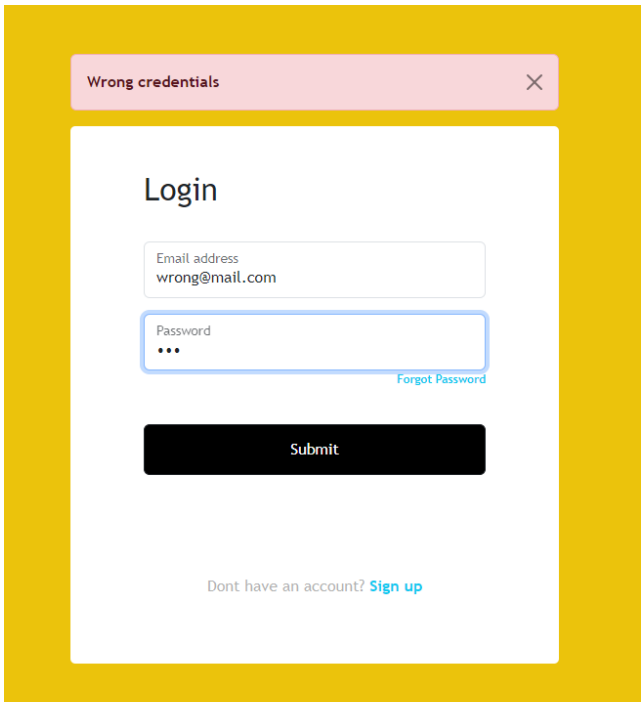
4.2. Σύνδεση

Αφού έχει εγγραφεί επιτυχώς, ο χρήστης μπορεί να συμπληρώσει τα στοιχεία της εγγραφής του για να συνδεθεί στην εφαρμογή.



111. Σελίδα σύνδεσης

Εάν τα στοιχεία δεν είναι σωστά, εμφανίζεται σχετικό ενημερωτικό μήνυμα. Αλλιώς, ο χρήστης μεταφέρεται στην αρχική σελίδα της εφαρμογής.

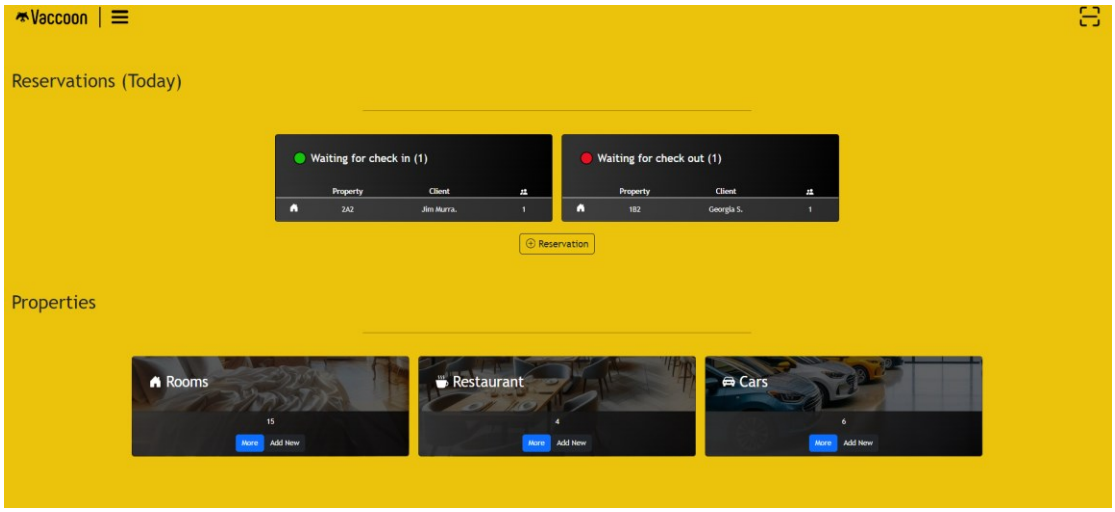


112. Παράδειγμα ανεπιτυχούς σύνδεσης

4.3. Αρχική σελίδα

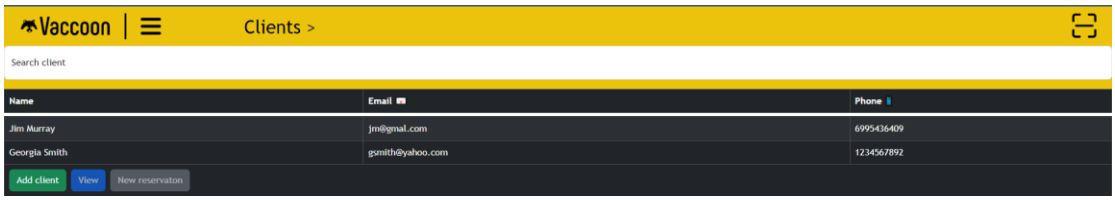
Στην αρχική σελίδα υπάρχει μια προεπισκόπηση της ημέρας με το ποιες κρατήσεις αναμένεται να γίνουν check in και check out σήμερα. Πατώντας πάνω σε κάποια από αυτές, μεταφερόμαστε στην σελίδα προβολής της κράτησης, όπου εκεί μπορούν να γίνουν περαιτέρω ενέργειες. Επίσης, βρίσκεται και ένα κουμπί για γρήγορη μεταφορά στην σελίδα δημιουργίας νέας κράτησης.

Παρακάτω έχουμε τρεις κάρτες για τις ιδιοκτησίες στις οποίες εμφανίζεται το πλήθος, κουμπί προβολής της λίστας των ιδιοκτησιών και κουμπί γρήγορης προσθήκης για κάθε είδος ξεχωριστά.



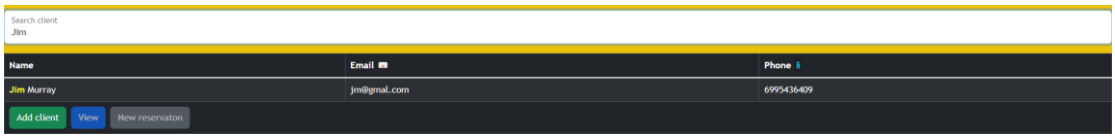
113. Αρχική Σελίδα

4.4. Σελίδα Πελατών



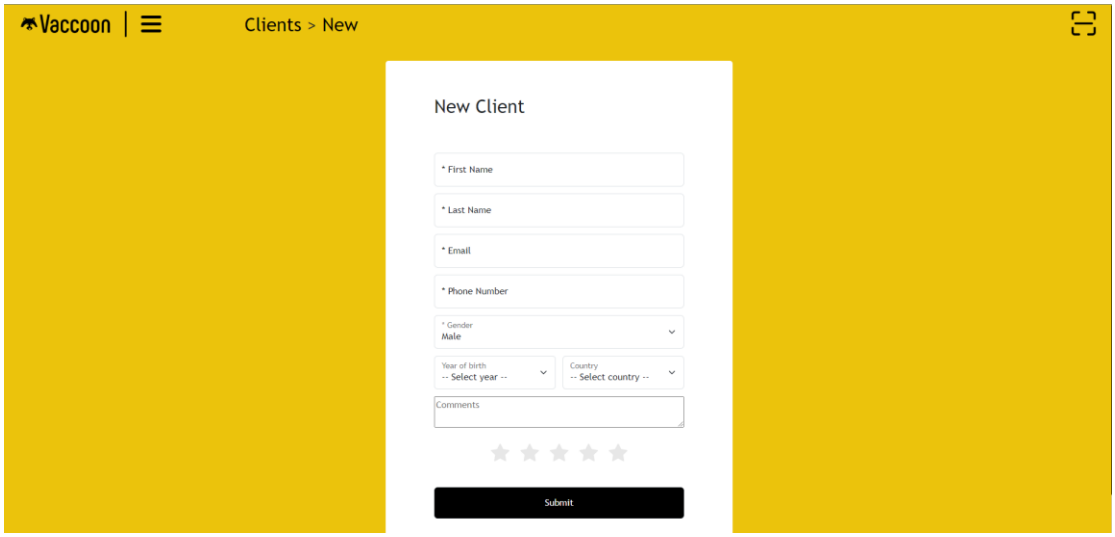
114. Σελίδα πελατών

Στην σελίδα πελατών, γίνεται προβολή της λίστας με όλους τους πελάτες του χρήστη. Στην κορυφή υπάρχει μπάρα αναζήτησης, για οποιοδήποτε προσωπικό στοιχείο, για ταχύτερη εύρεση του πελάτη.



115. Παράδειγμα φίλτρου αναζήτησης στη σελίδα πελατών

Κάτω από την λίστα βρίσκονται τρία κουμπιά. Το πρώτο κουμπί “Add Client” μας μεταφέρει στη φόρμα συμπλήρωσης προσθήκης νέου πελάτη. Τα κουμπιά “View” και “New reservation” είναι απενεργοποιημένα.



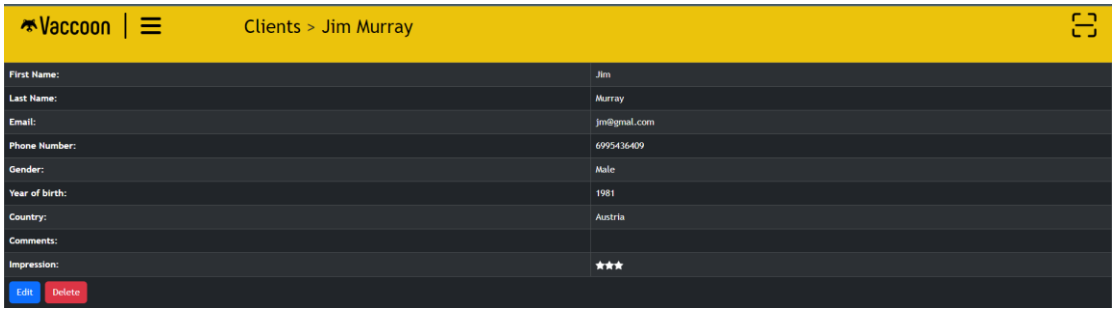
116. Σελίδα προσθήκης νέου πελάτη

Στη σελίδα προσθήκης του πελάτη, συμπληρώνουμε τα στοιχεία του, και σαν τελευταία επιλογή, είναι η επιλογή αστεριών που αντιστοιχεί στην εντύπωση που μας έχει αφήσει ο πελάτης. Τα πεδία χωρίς αστερίσκο (‘*’) δεν είναι υποχρεωτικά για συμπλήρωση.

Επιλέγοντας κάποιον πελάτη, η γραμμή του γίνεται highlight και ενεργοποιούνται τα κουμπιά “View” και “New reservation”.

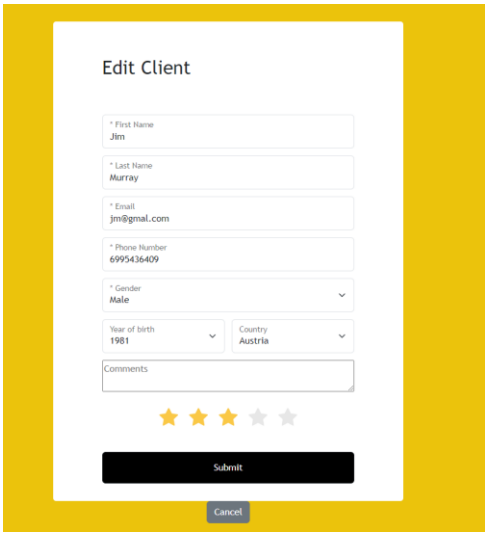
117. Επιλογή πελάτη

Πατώντας το κουμπί “View” μεταφερόμαστε στη σελίδα προβολής των στοιχείων του πελάτη.



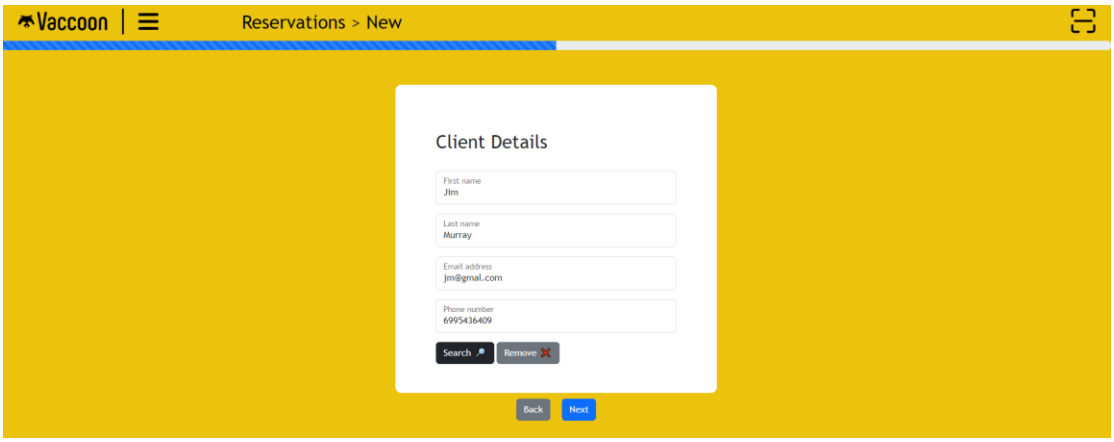
118. Σελίδα Πελάτη

Από εκεί μπορούμε να επεξεργαστούμε τα στοιχεία του, πατώντας στο κουμπί “Edit” όπου μας μεταφέρει στην φόρμα συμπλήρωσης στοιχείων με προσυμπληρωμένα τα υπάρχοντα, ή να τον διαγράψουμε πατώντας το κουμπί “Delete”.



119. Σελίδα Επεξεργασίας Πελάτη

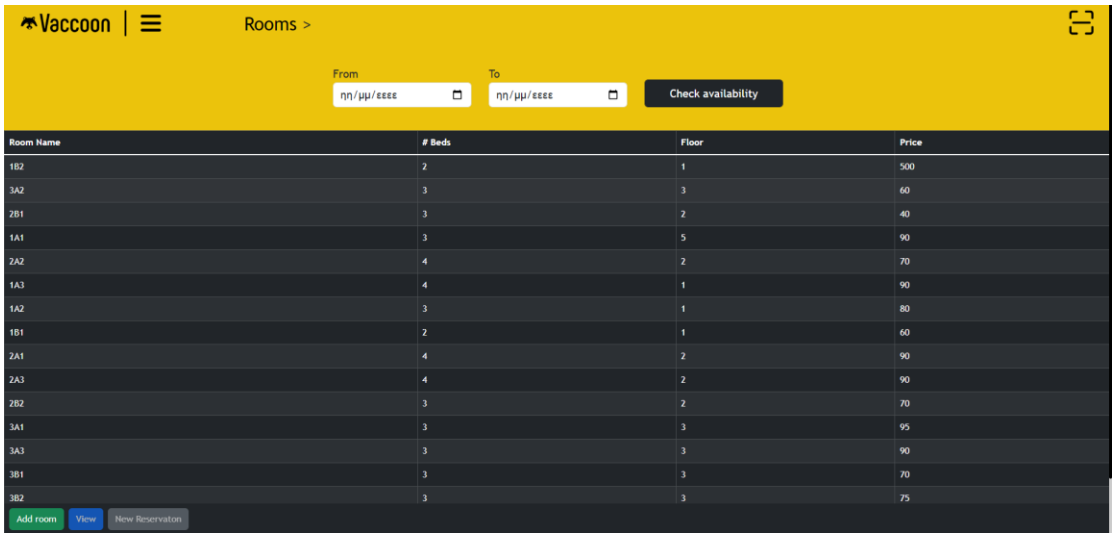
Πατώντας το κουμπί “New reservation” μεταφερόμαστε στη σελίδα προσθήκης νέας εγγραφής με προεπιλεγμένο τον πελάτη που έχουμε επιλέξει.



120. Σελίδα νέας κράτησης μέσω της σελίδας πελάτη

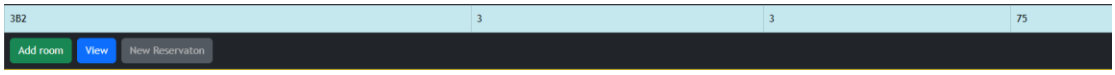
4.5. Σελίδα Δωματίων

Στην σελίδα δωματίων βρίσκουμε μία λίστα με όλα τα δωμάτια του χρήστη.



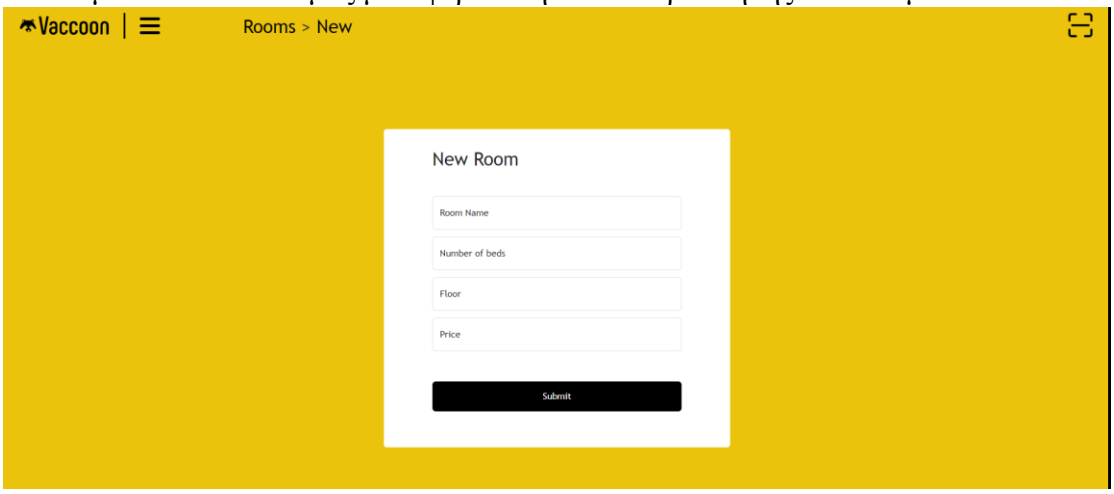
121. Σελίδα δωματίων

Επιλέγοντας κάποιο δωμάτιο, αυτό γίνεται highlight και ενεργοποιούνται τα κουμπιά "View" και "New reservation". Συγκεκριμένα για το "New Reservation" θα πρέπει να έχει επιλεχθεί το κουμπί "Check availability" **πρώτα** αλλιώς θα παραμείνει απενεργοποιημένο.



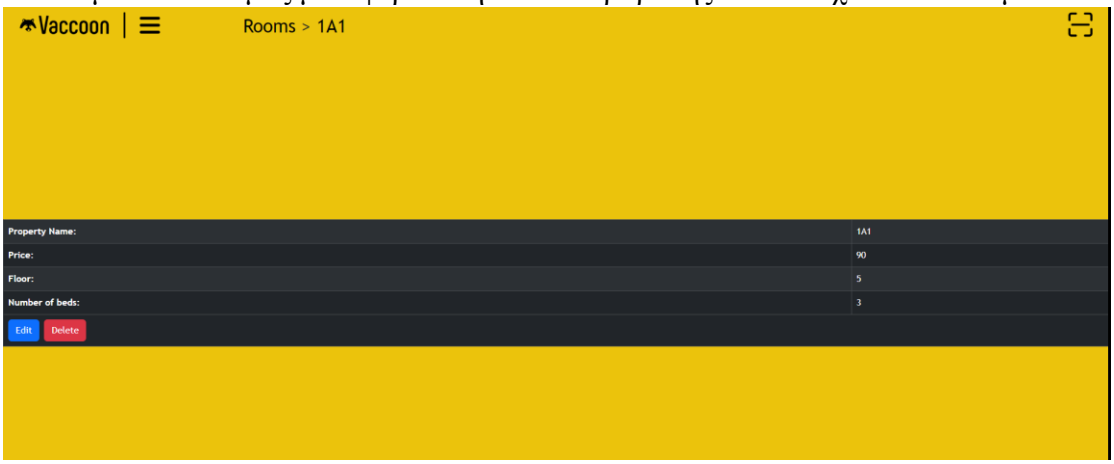
122. Επιλογή Δωματίου

Το κουμπί “Add room” μας μεταφέρει στη σελίδα προσθήκης νέου δωματίου.



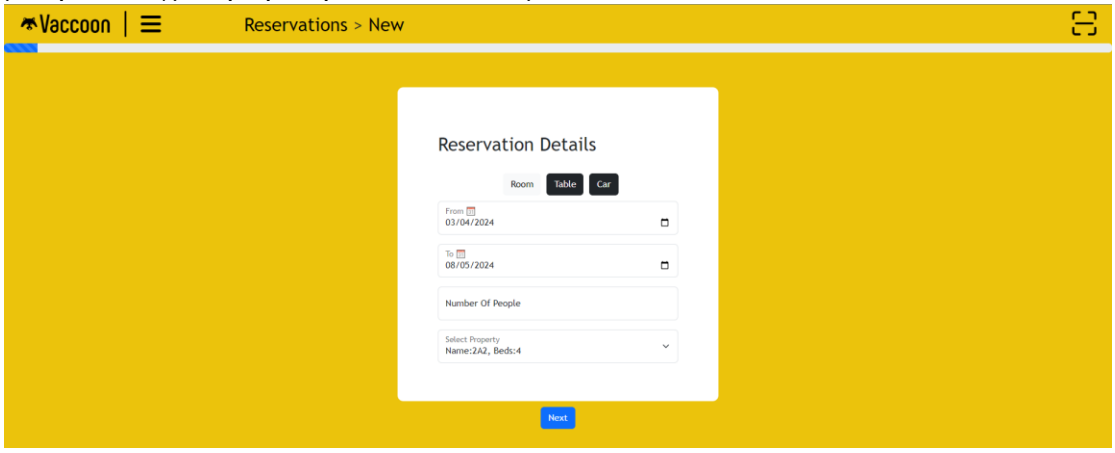
123. Σελίδα προσθήκης δωματίου

Το κουμπί “View” μας μεταφέρει στη σελίδα προβολής των στοιχείων του δωματίου.



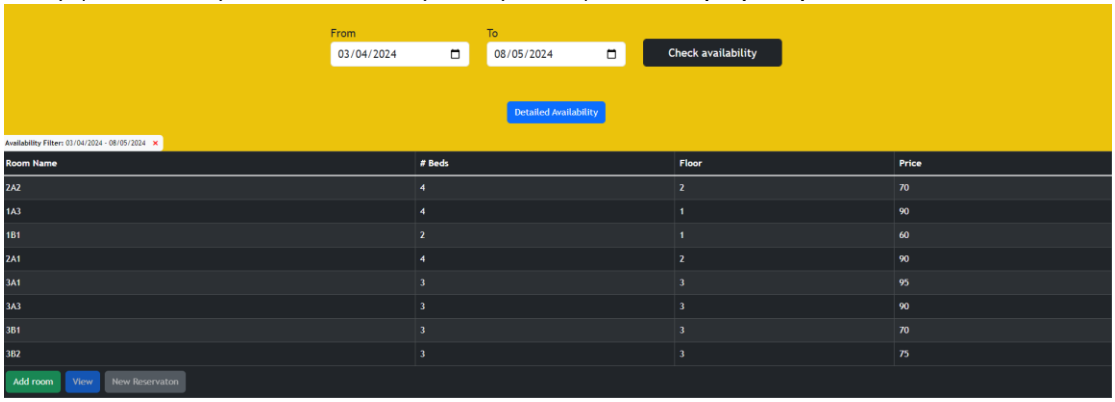
124. Σελίδα δωματίου

Το κουμπί “New reservation” μας μεταφέρει στη σελίδα δημιουργίας νέας κράτησης με προεπιλεγμένη την περίοδο και το δωμάτιο.



125. Σελίδα νέας κράτησης μέσω της σελίδας δωματίου

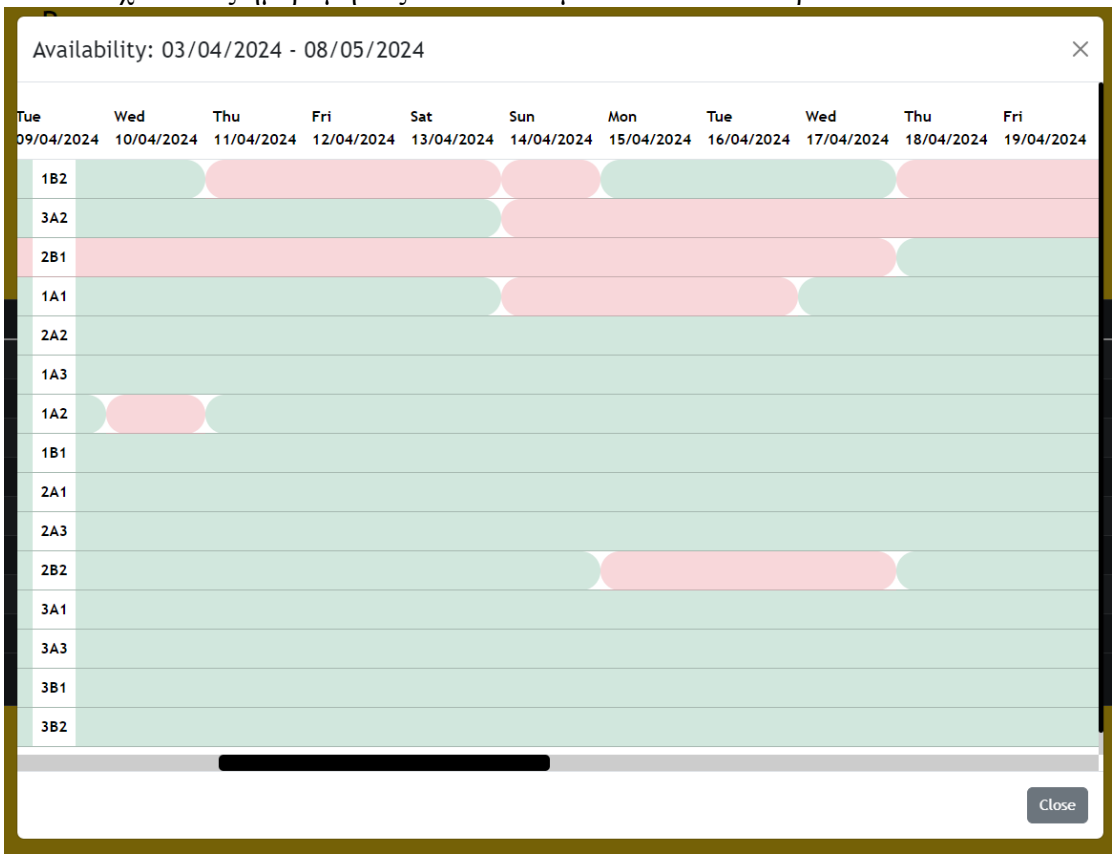
Στην κορυφή υπάρχουν τα πεδία αναζήτησης ημερομηνιών διαθεσιμότητας. Βάζοντας τις ημερομηνίες που μας ενδιαφέρουν και πατώντας “Check availability”, στην λίστα θα εμφανιστούν μόνο τα διαθέσιμα δωμάτια για εκείνη την περίοδο.



126. Φίλτρο περιόδου διαθεσιμότητας δωματίων

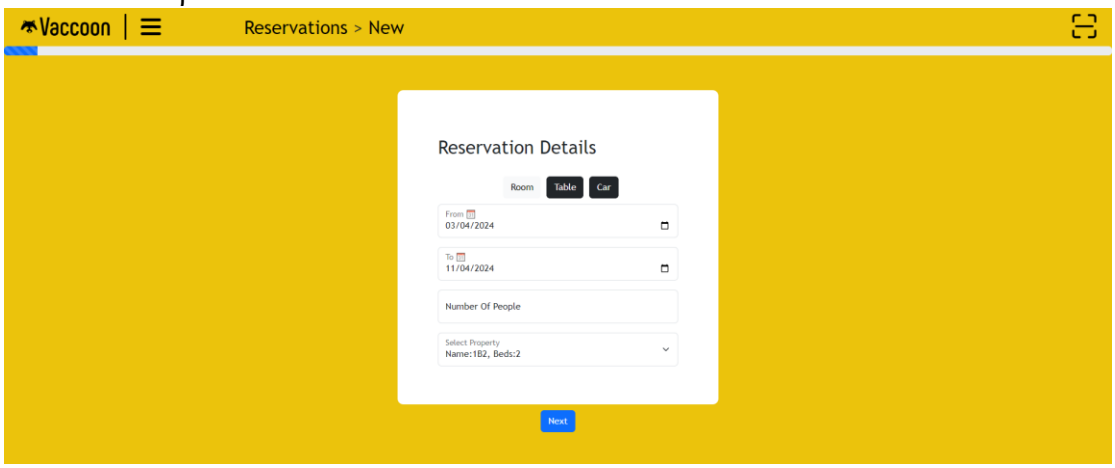
Αφού πατηθεί το κουμπί “Check availability” εμφανίζεται το κουμπί “**Detailed Availability**”. Πατώντας το, εμφανίζεται ένα παράθυρο με αναλυτική προβολή της διαθεσιμότητας όλων των ημερών της επιλεγμένης περιόδου. Τα κόκκινα πεδία δείχνουν τις ημερομηνίες τις οποίες υπάρχει κράτηση στο δωμάτιο, ενώ τα πράσινα

πεδία δείχνουν τις ημερομηνίες όπου το δωμάτιο είναι ελεύθερο.



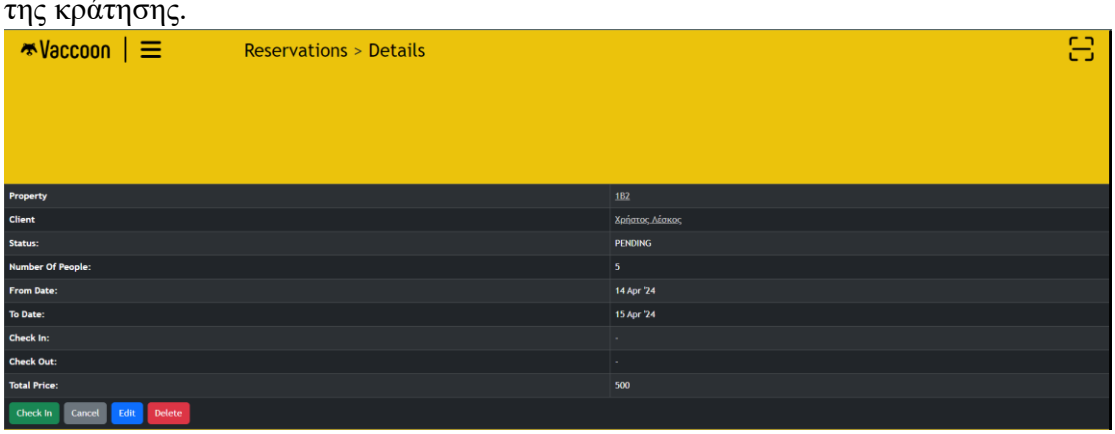
127. Αναλυτική προβολή διαθεσιμότητας δωματίων

Πατώντας σε κάποιο από τα **πράσινα πεδία**, γίνεται μεταφορά στην σελίδα δημιουργίας νέας κράτησης με προεπιλεγμένο το δωμάτιο και την περίοδο που καλύπτει το πράσινο πεδίο.



128. Σελίδα νέας κράτησης μέσω της αναλυτικής διαθεσιμότητας δωματίων

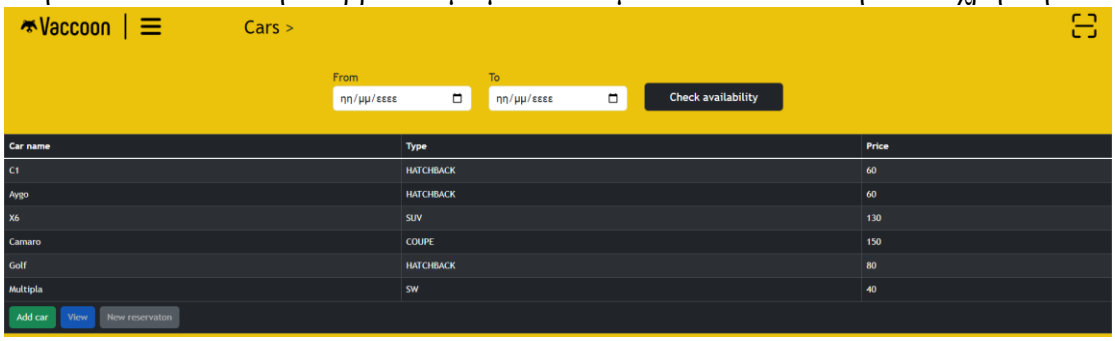
Πατώντας σε κάποιο από τα **κόκκινα πεδία**, γίνεται μεταφορά στη σελίδα ποβολής της κράτησης.



129. Σελίδα κράτησης μέσω της αναλυτικής διαθεσιμότητας δωματίων

4.6. Σελίδα Αυτοκινήτων

Στην σελίδα αυτοκινήτων βρίσκουμε μία λίστα με όλα τα αυτοκίνητα του χρήστη.

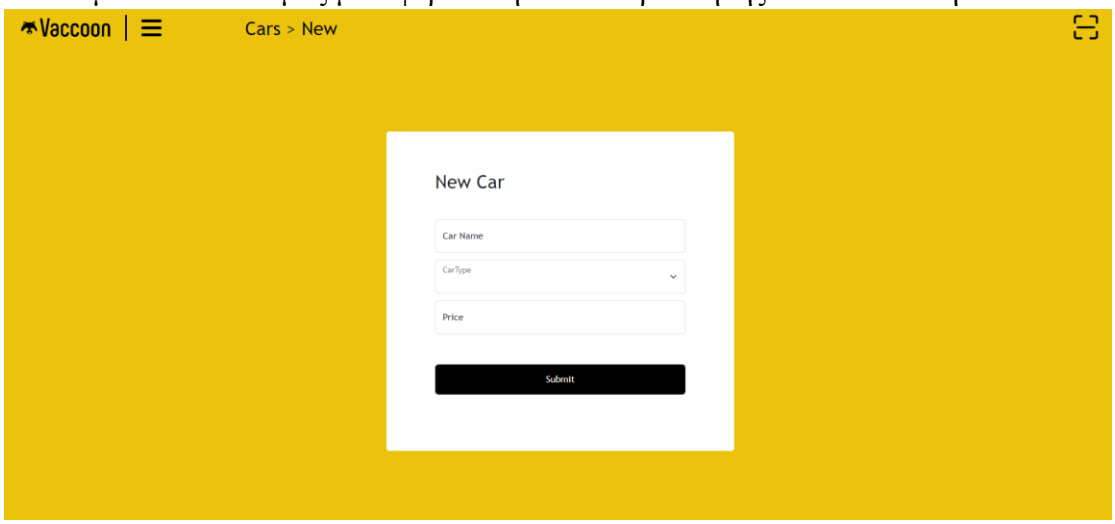


130. Σελίδα αυτοκινήτων

Επιλέγοντας κάποιο αυτοκίνητο, αυτό γίνεται highlight και ενεργοποιούνται τα κουμπιά "View" και "New reservation". Συγκεκριμένα για το "New Reservation" θα πρέπει να έχει επιλεχθεί το κουμπί "Check availability" **πρώτα** αλλιώς θα παραμείνει απενεργοποιημένο.

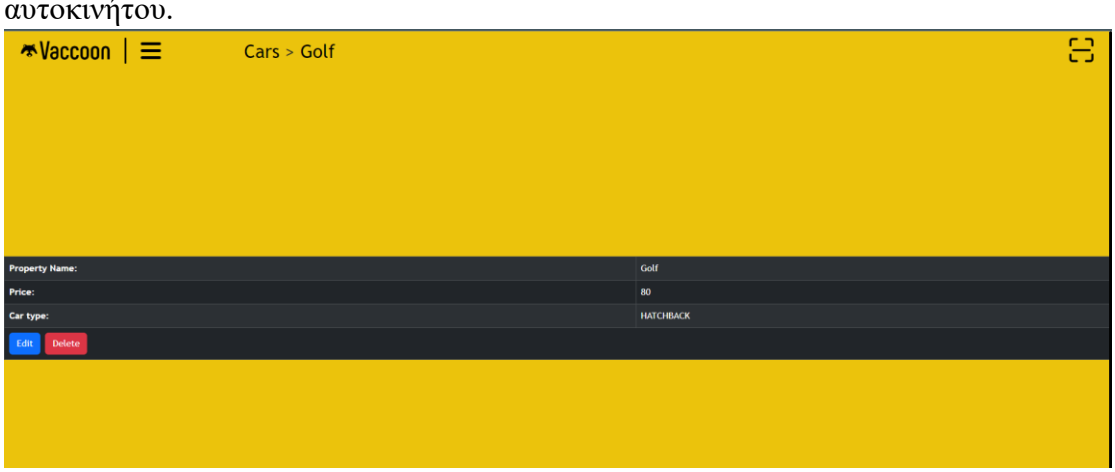
131. Επιλογή αυτοκινήτου

Το κουμπί "Add car" μας μεταφέρει στη σελίδα προσθήκης νέου αυτοκινήτου.



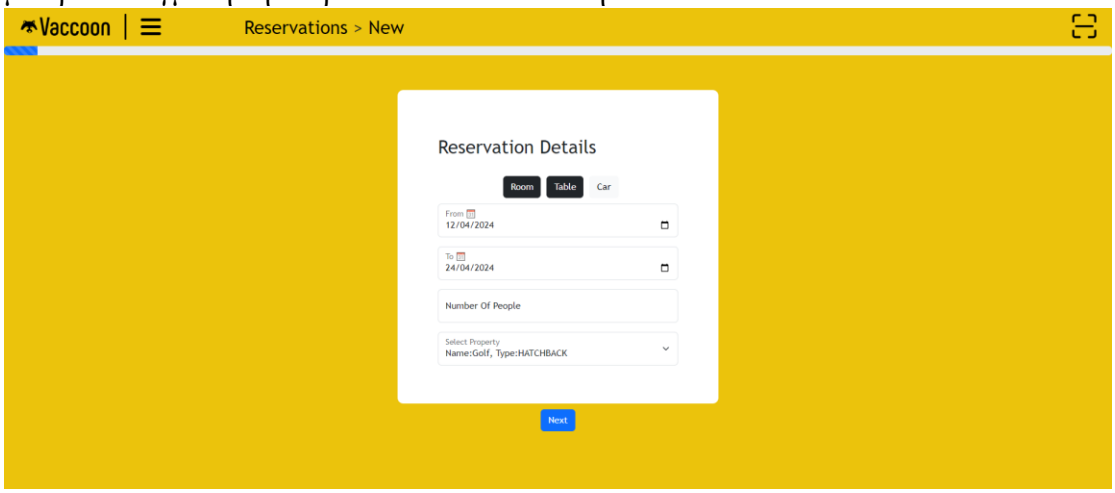
132. Σελίδα προσθήκης νέου αυτοκινήτου

Το κουμπί “View” μας μεταφέρει στη σελίδα προβολής των στοιχείων του αυτοκινήτου.



133. Σελίδα αυτοκινήτου

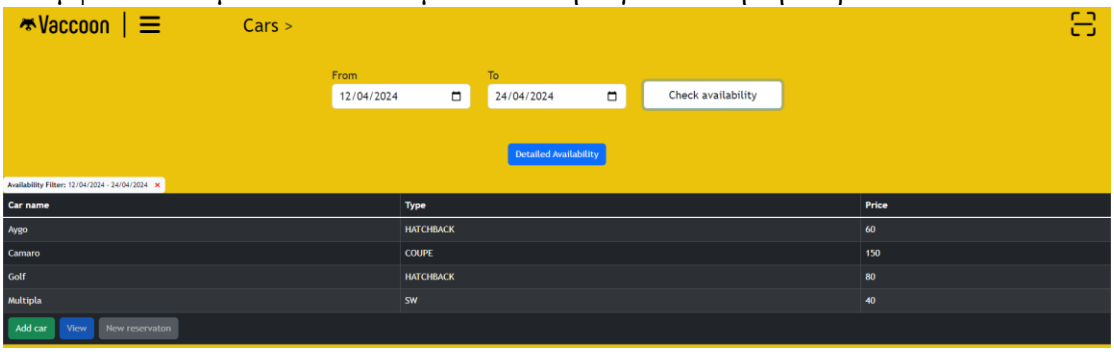
Το κουμπί “New reservation” μας μεταφέρει στη σελίδα δημιουργίας νέας κράτησης με προεπιλεγμένη την περίοδο και το αυτοκίνητο.



134. Σελίδα νέας κράτησης μέσω της σελίδας αυτοκινήτου

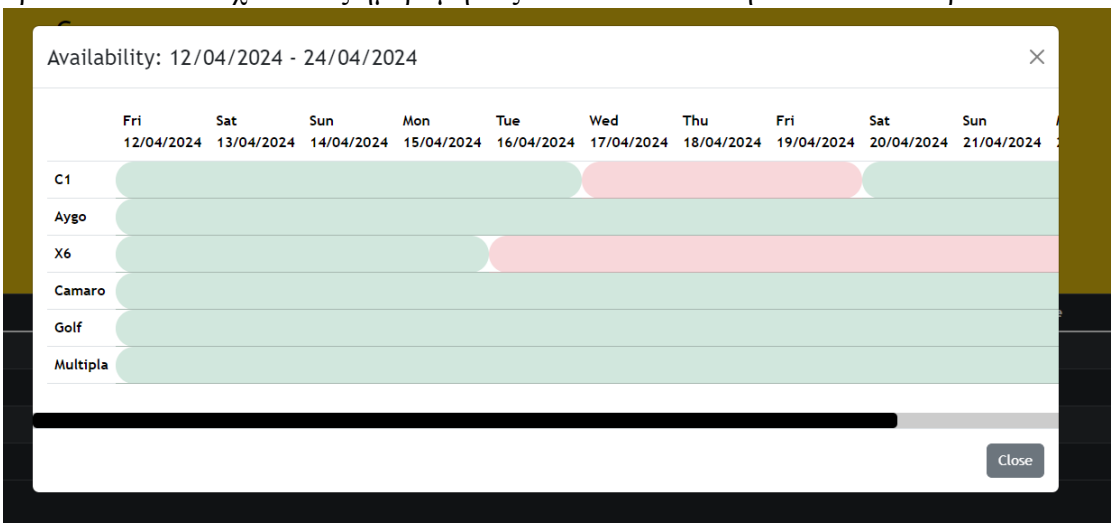
Στην κορυφή υπάρχουν τα πεδία αναζήτησης ημερομηνιών διαθεσιμότητας. Βάζοντας τις ημερομηνίες που μας ενδιαφέρουν και πατώντας “Check availability”, στην λίστα

θα εμφανιστούν μόνο τα διαθέσιμα αυτοκίνητα για εκείνη την περίοδο.



135. Φίλτρο περιόδου διαθεσιμότητας αυτοκινήτων

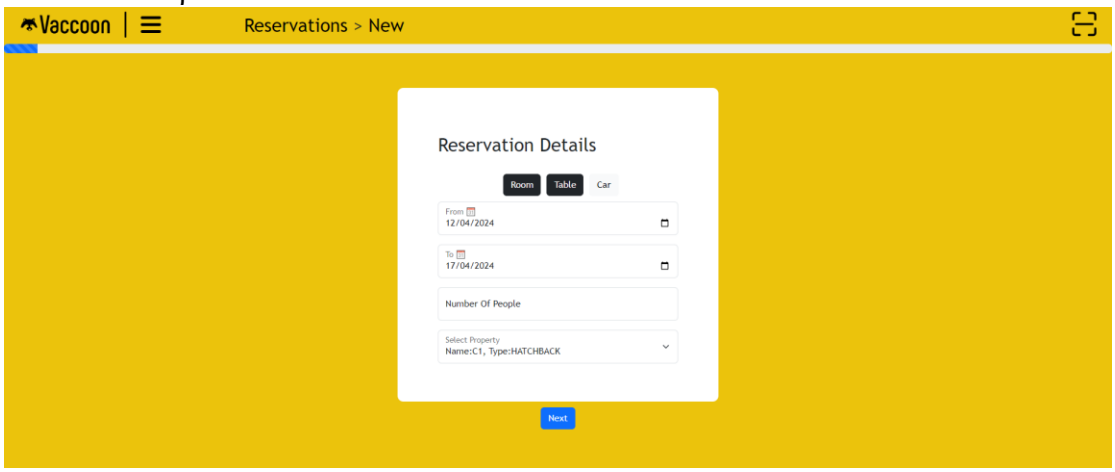
Αφού πατηθεί το κουμπί "Check availability" εμφανίζεται το κουμπί "**Detailed Availability**". Πατώντας το, εμφανίζεται ένα παράθυρο με αναλυτική προβολή της διαθεσιμότητας όλων των ημερών της επιλεγμένης περιόδου. Τα κόκκινα πεδία δείχνουν τις ημερομηνίες τις οποίες υπάρχει κράτηση στο αυτοκίνητο, ενώ τα πράσινα πεδία δείχνουν τις ημερομηνίες όπου το αυτοκίνητο είναι ελεύθερο.



136. Αναλυτική διαθεσιμότητα αυτοκινήτων

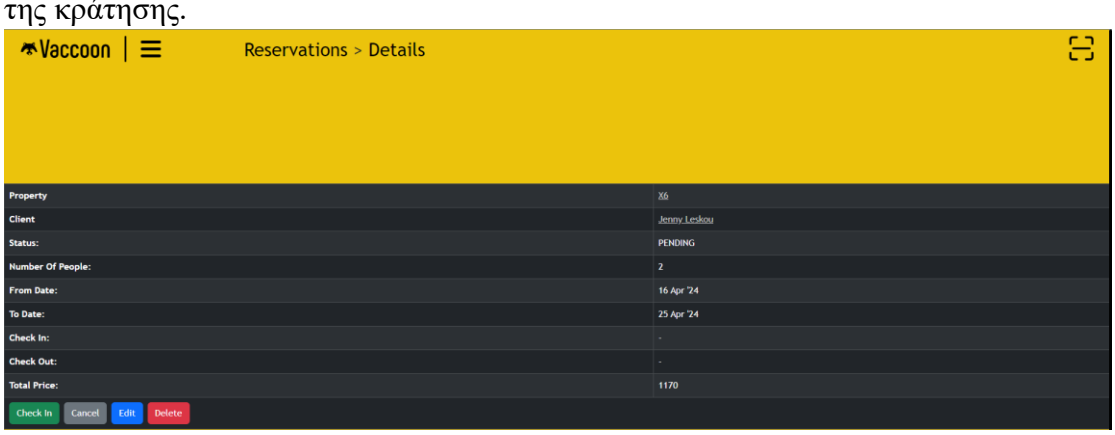
Πατώντας σε κάποιο από τα **πράσινα πεδία**, γίνεται μεταφορά στην σελίδα δημιουργίας νέας κράτησης με προεπιλεγμένο το αυτοκίνητο και την περίοδο που

καλύπτει το πράσινο πεδίο.



137. Σελίδα νέας κράτησης μέσω της αναλυτικής διαθεσιμότητας αυτοκινήτων

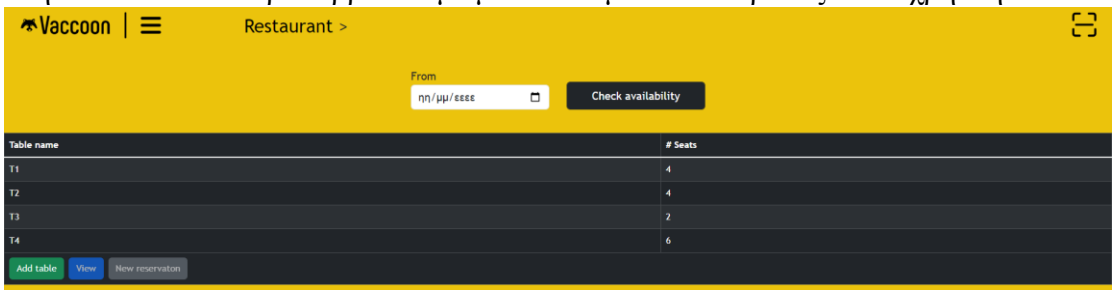
Πατώντας σε κάποιο από τα **κόκκινα πεδία**, γίνεται μεταφορά στη σελίδα ποβολής της κράτησης.



138. Σελίδα κράτησης μέσω της αναλυτικής διαθεσιμότητας αυτοκινήτων

4.7. Σελίδα Εστιατορίου

Στην σελίδα εστιατορίου βρίσκουμε μία λίστα με όλα τα τραπέζια του χρήστη.

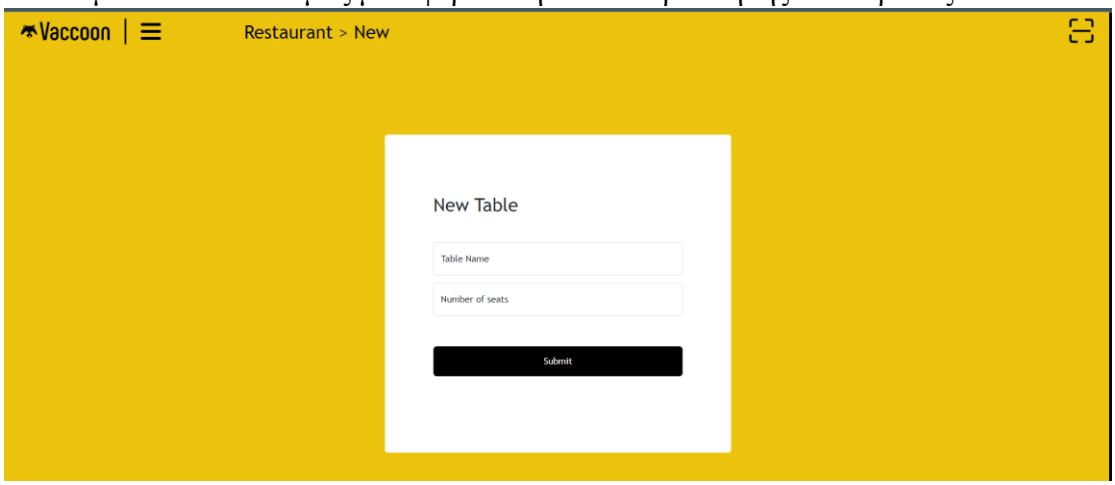


139. Σελίδα εστιατορίου

Επιλέγοντας κάποιο τραπέζι, αυτό γίνεται highlight και ενεργοποιούνται τα κουμπιά "View" και "New reservation". Συγκεκριμένα για το "New Reservation" θα πρέπει να έχει επιλεχθεί το κουμπί "Check availability" **πρώτα** αλλιώς θα παραμείνει απενεργοποιημένο.

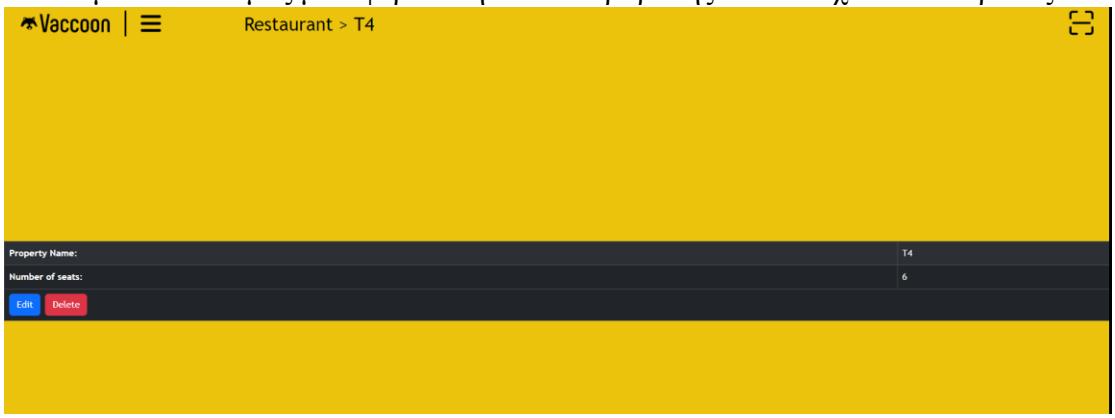
140. Επιλογή τραπεζιού

Το κουμπί "Add table" μας μεταφέρει στη σελίδα προσθήκης νέου τραπεζιού.



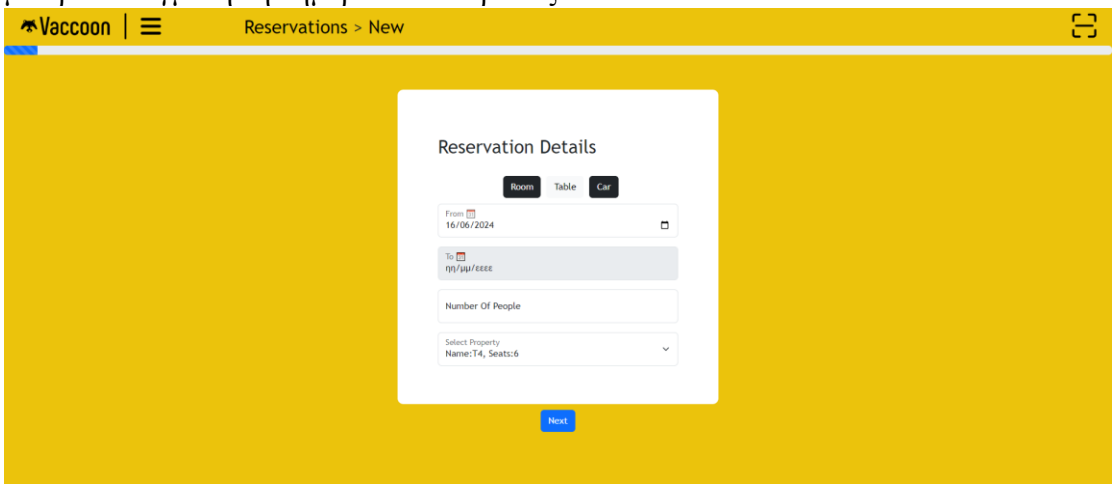
141. Σελίδα προσθήκης νέου τραπεζιού

Το κουμπί “View” μας μεταφέρει στη σελίδα προβολής των στοιχείων του τραπεζιού.



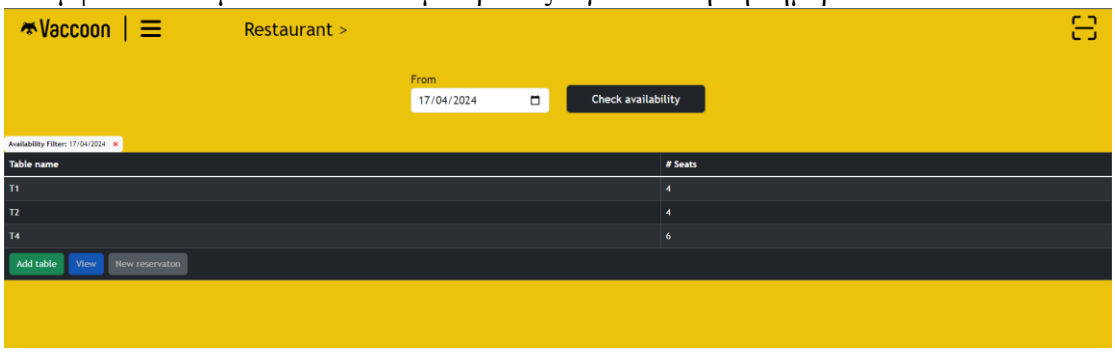
142. Σελίδα τραπεζιού

Το κουμπί “New reservation” μας μεταφέρει στη σελίδα δημιουργίας νέας κράτησης με προεπιλεγμένη την ημέρα και το τραπέζι.



143. Σελίδα νέας κράτησης μέσω της σελίδας τραπεζιού

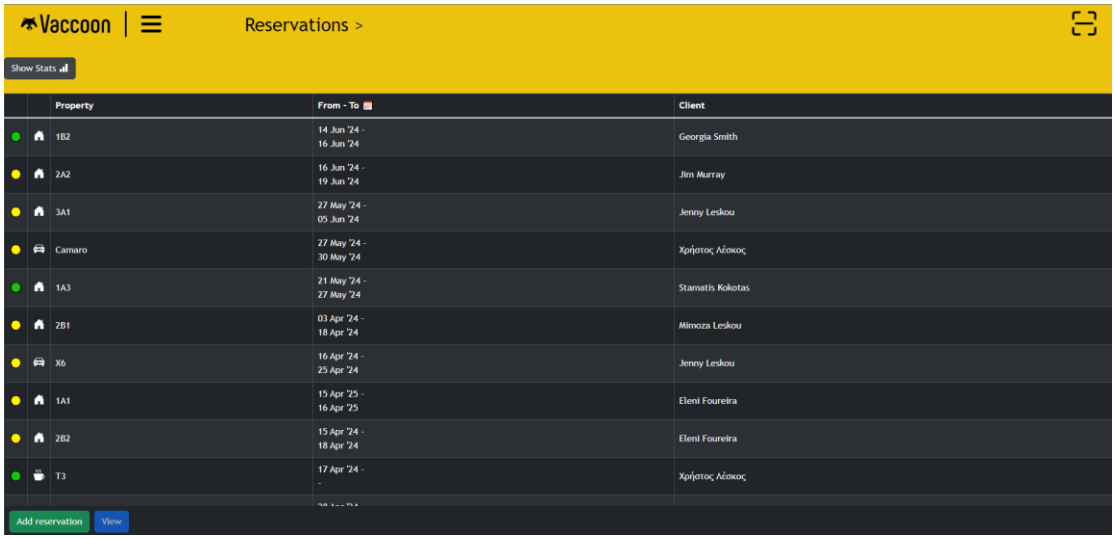
Στην κορυφή υπάρχει το πεδίο αναζήτησης ημερομηνίας διαθεσιμότητας. Βάζοντας την ημερομηνία που μας ενδιαφέρει και πατώντας “Check availability”, στην λίστα θα εμφανιστούν μόνο τα διαθέσιμα τραπέζια για εκείνη την ημέρα.



144. Φίλτρο ημερομηνίας διαθεσιμότητας τραπεζιών

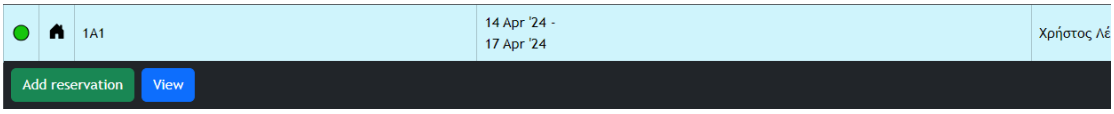
4.8. Σελίδα Κρατήσεων

Στην σελίδα κρατήσεων, εμφανίζεται μία λίστα με όλες τις κρατήσεις του χρήστη.



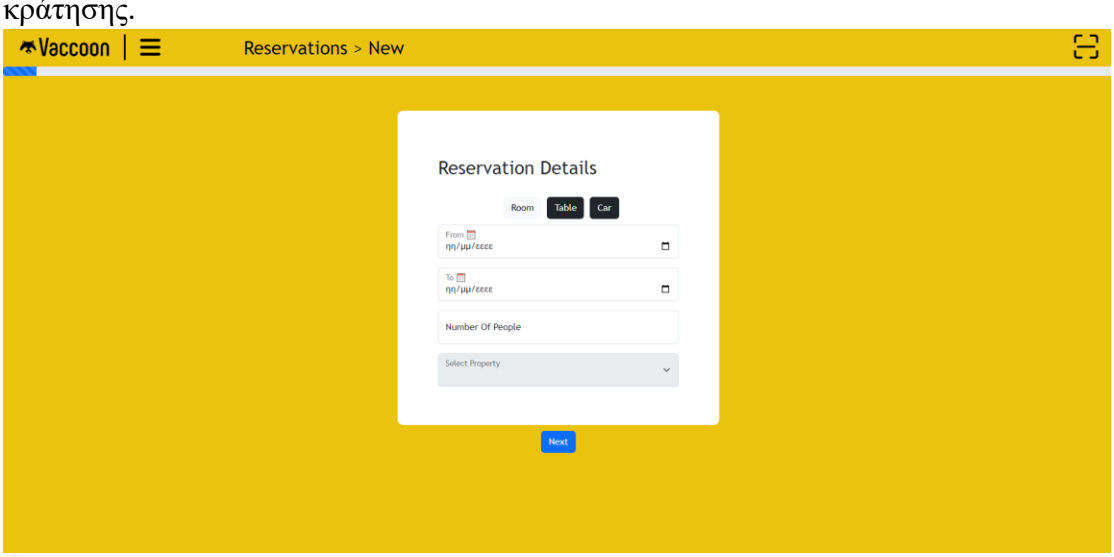
145. Σελίδα κρατήσεων

Επιλέγοντας μία κράτηση, αυτή γίνεται highlight και ενεργοποιείται το κουμπί “View”.



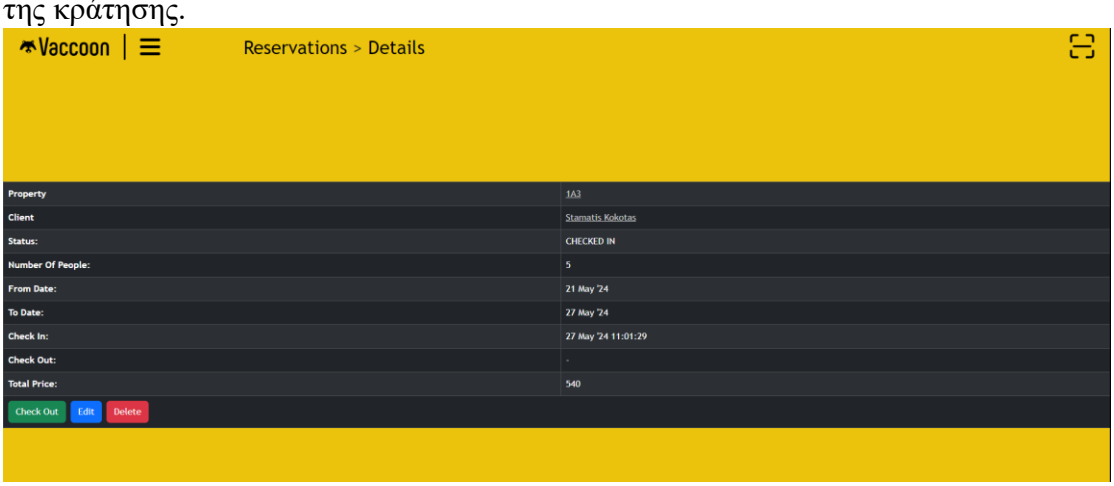
146. Επιλογή κράτησης

Πατώντας το κουμπί “Add reservation” μεταφερόμαστε στην σελίδα προσθήκης νέας κράτησης.



147. Σελίδα προσθήκης νέας κράτησης

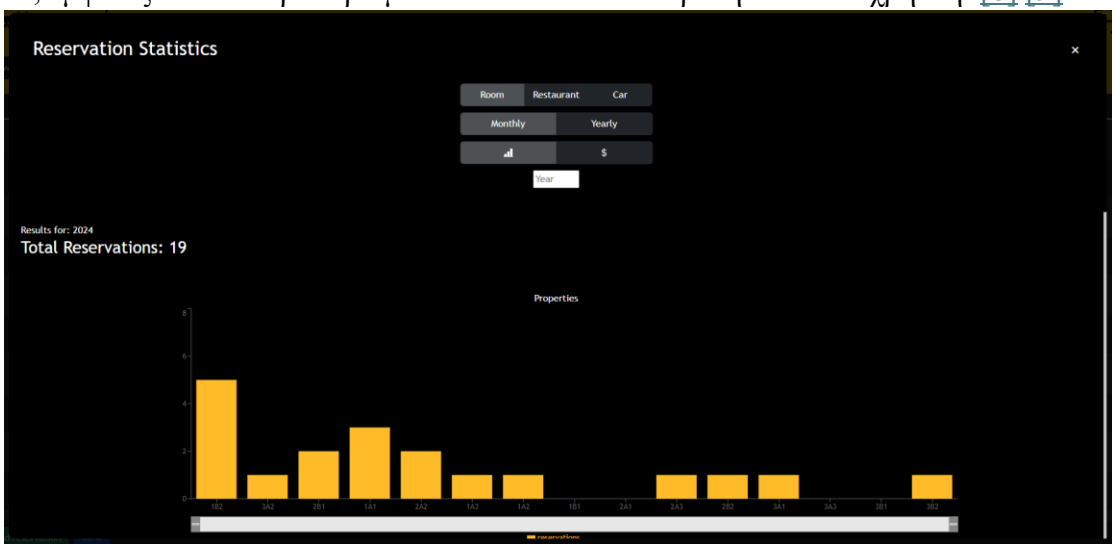
Πατώντας το κουμπί “View”, μεταφερόμαστε στην σελίδα προβολής των στοιχείων της κράτησης.



148. Σελίδα κράτησης

Εκεί μπορούμε να κάνουμε check in, checkout ή cancel την κράτηση να την επεξεργαστούμε και να την διαγράψουμε.

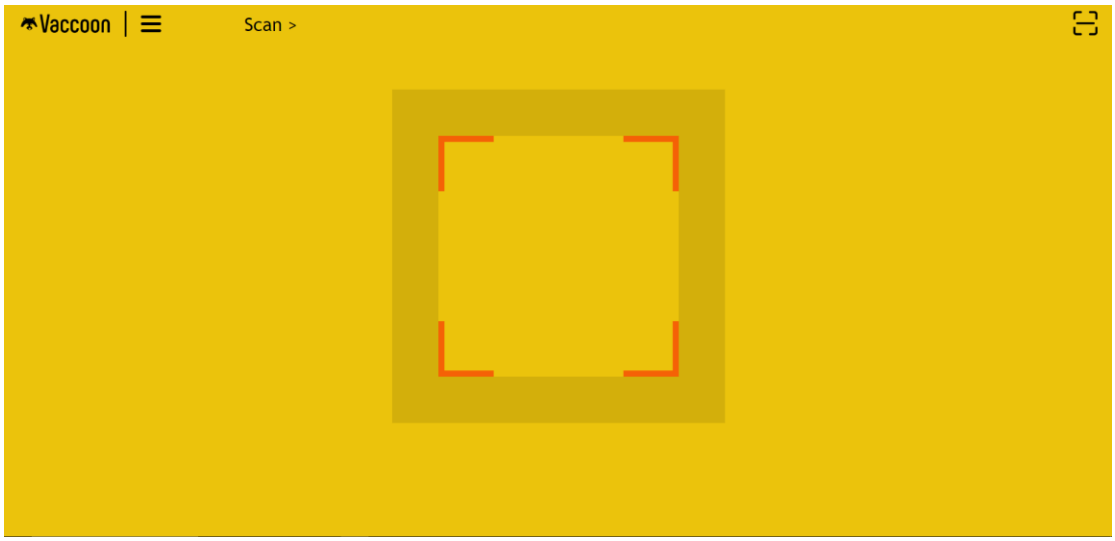
Στην κορυφή της σελίδας κρατήσεων, βρίσκεται το κουμπί “Show Stats”. Πατώντας το, εμφανίζεται το παράθυρο με τα στατιστικά των κρατήσεων του χρήστη. [8] [9]



149. Παράθυρο στατιστικών κρατήσεων

4.9. Σελίδα Σάρωσης

Στη σελίδα σάρωσης, χρησιμοποιείται η κάμερα της συσκευής του χρήστη. Σαρώνει το QR code [10] της κράτησης του πελάτη και εάν γίνει επιτυχώς, μας μεταφέρει στη σελίδα πληροφοριών της κράτησης. Εκεί μπορούμε να κάνουμε check in, check out, cancel, διαγραφή ή να επεξεργαστούμε την κράτηση.



150. Σελίδα σάρωσης κράτησης

5. Συμπεράσματα και Επόμενα Βήματα

Η οργάνωση της κατάστασης των κρατήσεων σε μια τουριστική επιχείρηση, μπορεί να απλουστεύσει την διαχείριση τους περιορίζοντας το άγχος του ιδιοκτήτη και τις πιθανότητες λάθους. Μία καλύτερη εικόνα μπορεί να οδηγήσει σε καλύτερες αποφάσεις και ενέργειες τόσο βραχυπρόθεσμα όσο και μακροπρόθεσμα. Το Vaccoon συμβάλει σε αυτό προσφέροντας όλες τις απαραίτητες λειτουργίες και πληροφορίες που θα χρειαστεί ο χρήστης για καλύτερο προγραμματισμό.

Στο μέλλον, το λογισμικό αυτό θα μπορούσε να επεκτείνει την χρηστικότητά του, επικοινωνώντας με εφαρμογές online κρατήσεων. Αυτό θα συντόνιζε τις online κρατήσεις μεταξύ διαφορετικών εφαρμογών τις οποίες ο ιδιοκτήτης χρησιμοποιεί για τις ιδιοκτησίες του και των απευθείας κρατήσεων στην επιχείρηση. Για παράδειγμα, ένας ιδιοκτήτης δωματίων ο οποίος έχει διαθέσει τα δωμάτιά του στην εφαρμογή του Booking.com, με κάθε online κράτηση που δέχεται, το Vaccoon θα ενημερώνεται για αυτή και θα ενημερώνει την κατάσταση. Αντίστοιχα, για κάθε φυσική κράτηση, το Vaccoon θα μπορεί να αφαιρεί την διαθεσιμότητα του δωματίου από την εφαρμογή. Συλλέγοντας επίσης σαν πληροφορία την πηγή της κράτησης και σαρώνοντας το QR code του πελάτη κατά τη άφιξή του, θα υπάρχει εύκολα πλήρης γνώση για όλες τις απαραίτητες λεπτομέρειες που χρειάζονται.

Βιβλιογραφία

1. Spring Boot Documentation (https://spring.io/projects/spring-boot)
2. Spring Security Documentation (https://spring.io/projects/spring-security)
3. JPA Query Methods Documentation (https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html)
4. JSON Recursive Return Bug fixed with @JsonIgnore (https://stackoverflow.com/questions/32298722/how-do-i-get-jsonignore-to-work-so-that-json-are-not-returned-recursively)
5. Bootstrap Docs (https://getbootstrap.com/docs/5.3/)
6. Bootstrap Icons (https://icons.getbootstrap.com/)
7. React Documentation (https://react.dev/learn)
8. Recharts graphs installation (https://recharts.org/en-US/guide)
9. Recharts graph types examples (https://recharts.org/en-US/examples)
10. Yudiel React QR Scanner installation and usage example (https://www.npmjs.com/package/@yudiel/react-qr-scanner)
11. How to use Axios with React (https://www.digitalocean.com/community/tutorials/react-axios-react)