

ΠΑΝΕΠΙΣΤΉΜΙΟ ΙΩΑΝΝΙΝΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΉΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΉΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΔΗΜΙΟΥΡΓΊΑ ΣΥΣΤΗΜΑΤΟΣ ΚΡΑΤΗΣΕΩΝ ΠΕΛΑΤΩΝ ΓΙΑ ΤΟΥΡΙΣΤΙΚΑ ΚΑΤΑΛΥΜΑΤΑ

ΚΩΝΣΤΑΝΤΙΝΟΣ ΖΕΡΑΣ

Επιβλέπων Καθηγητής

Νικόλαος Γιαννακέας

Άρτα, Ιούλιος 2026

CUSTOMER BOOKING SYSTEM FOR TOURIST ACCOMMODATIONS

Εγκρίθηκε από τριμελή εξεταστική επιτροπή

Άρτα, 15/09/2026

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Επιβλέπων καθηγητής

   Νικόλαος Γιαννακέας,

   Αναπληρωτής Καθηγητής

2. Μέλος επιτροπής

   Αλέξανδρος Τζάλλας,

   Αναπληρωτής Καθηγητής

3. Μέλος επιτροπής

   Δημήτριος Δημόπουλος,

   Ακαδημαϊκός Υπότροφος

© ΖΕΡΑΣ ΚΩΝΣΤΑΝΤΙΝΟΣ, 2026

Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας ότι η παρούσα μεταπτυχιακή εργασία είναι εξ ολοκλήρου αποτέλεσμα δικής μου ερευνητικής εργασίας δεν αποτελεί προϊόν αντιγραφής ούτε προέρχεται από ανάθεση σε τρίτους. Όλες οι πηγές που χρησιμοποιήθηκαν (κάθε είδους, μορφής και προέλευσης) για τη συγγραφή της περιλαμβάνονται στην βιβλιογραφία.

Ζέρας Κωνσταντίνος,

Υπογραφή

Περίληψη

Η παρούσα πτυχιακή εργασία έχει ως αντικείμενο τη σχεδίαση, την υλοποίηση και την αξιολόγηση ενός ολοκληρωμένου συστήματος κρατήσεων πελατών για τουριστικά καταλύματα μικρής κλίμακας. Το σύστημα αναπτύχθηκε ως διαδικτυακή εφαρμογή με το πλαίσιο Django της γλώσσας Python, ακολουθώντας την αρχιτεκτονική Model-View-Template και την αρχή του διαχωρισμού ευθυνών, με την κρίσιμη επιχειρησιακή λογική συγκεντρωμένη σε αυτόνομο επίπεδο υπηρεσιών (services) και όχι διάσπαρτη στις προβολές (views) της εφαρμογής.

Η εργασία εστιάζει στον πυρήνα του προβλήματος των κρατήσεων, δηλαδή στον ακριβή υπολογισμό της διαθεσιμότητας και στην αποτροπή επικαλυπτόμενων κρατήσεων. Η διαθεσιμότητα αντιμετωπίζεται ως δυναμικό αποτέλεσμα επιχειρησιακών κανόνων και όχι ως στατική αποθηκευμένη πληροφορία: υπολογίζεται εκ νέου από τις εγκεκριμένες κρατήσεις, τις περιόδους μη διαθεσιμότητας και τα εκκρεμή αιτήματα που διατηρούν ενεργή προσωρινή δέσμευση (hold). Υιοθετείται η κλειστή–ανοικτή σημασιολογία χρονικών διαστημάτων [check_in, check_out), ώστε διαδοχικές διαμονές να μη θεωρούνται συγκρουόμενες. Η ροή «αίτημα – απόφαση» εκφράζεται μέσω ρητών καταστάσεων κράτησης και εκτελείται εντός συναλλαγών βάσης δεδομένων με κλείδωμα εγγραφών, ώστε η ακεραιότητα των δεδομένων να διατηρείται και υπό συνθήκες ταυτόχρονης πρόσβασης.

Το σύστημα υποστηρίζει τρεις ρόλους χρηστών (επισκέπτης, εγγεγραμμένος πελάτης και ιδιοκτήτης/διαχειριστής) και επιβάλλει την εξουσιοδότηση τόσο σε επίπεδο προβολών, μέσω decorators ελέγχου ρόλου, όσο και σε επίπεδο ερωτημάτων προς τη βάση δεδομένων, ώστε ένας χρήστης να μην μπορεί να προσπελάσει εγγραφές που δεν του ανήκουν. Παρέχεται, επιπλέον, πίνακας διαχείρισης για την έγκριση ή απόρριψη αιτημάτων, τη διαχείριση αιτημάτων ακύρωσης και τροποποίησης και τη δημιουργία περιόδων μη διαθεσιμότητας, ενώ κάθε κρίσιμη ενέργεια καταγράφεται σε ξεχωριστή οντότητα ιστορικού, εξασφαλίζοντας δυνατότητα εκ των υστέρων ελέγχου.

Η αξιολόγηση πραγματοποιήθηκε με αντιστοίχιση των διατυπωμένων απαιτήσεων στα αντίστοιχα σημεία της υλοποίησης, καθώς και με συγκριτική εξέταση ανταγωνιστικών εμπορικών και ανοικτού κώδικα λύσεων. Τα αποτελέσματα δείχνουν ότι η εφαρμογή καλύπτει

πλήρως τον βασικό κύκλο ζωής μιας κράτησης και συγκροτεί τεκμηριωμένη βάση για μελλοντικές επεκτάσεις, όπως η ενσωμάτωση πληρωμών και η διασύνδεση με εξωτερικά κανάλια διάθεσης.

Λέξεις κλειδιά: σύστημα κρατήσεων, τουριστικά καταλύματα, Django, Python, διαθεσιμότητα, επικάλυψη ημερομηνιών, προσωρινή δέσμευση, συναλλαγές βάσης δεδομένων, έλεγχος πρόσβασης, UML

Abstract

This thesis addresses the design, implementation and evaluation of a complete customer booking system for small-scale tourist accommodations. The system is a web application built with the Django framework of the Python language, following the Model-View-Template architecture and a strict separation of concerns, with the critical business logic concentrated in a dedicated service layer rather than scattered across the views of the application.

The work focuses on the core of the booking problem, namely the accurate computation of availability and the prevention of overlapping reservations. Availability is treated as the dynamic outcome of business rules rather than as stored, static information: it is recomputed from approved bookings, blocked periods and pending requests that still carry an active temporary hold. The system adopts half-open interval semantics, [check_in, check_out), so that consecutive stays are not reported as conflicts. The request-and-decision workflow is expressed through explicit booking states and is executed inside database transactions with row-level locking, so that data integrity is preserved under concurrent access.

Three user roles are supported (visitor, registered customer and owner/administrator) and authorisation is enforced both at the view level, through role-checking decorators, and at the query level, so that a user cannot reach records that do not belong to them. An administrative dashboard supports the approval and rejection of requests, the handling of cancellation and modification requests and the creation of blocked periods, while every critical action is recorded in a separate audit-log entity, allowing decisions to be reviewed afterwards.

The evaluation was carried out by mapping the stated requirements onto the corresponding implementation points and by comparing the system with competing commercial and open-source solutions. The results show that the application fully covers the basic life cycle of a reservation and provides a documented basis for future extensions, such as payment integration and connectivity with external distribution channels.

Keywords: booking system, tourist accommodation, Django, Python, availability, date overlap, temporary hold, database transactions, access control, UML

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

Δήλωση μη λογοκλοπής 5

Περίληψη 6

Abstract 8

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ 9

Πίνακας Εικόνων 13

1. ΕΙΣΑΓΩΓΗ 14

1.1 Πλαίσιο και Αντικείμενο της Εργασίας 14

1.2 Σκοπός και Στόχοι της Εργασίας 15

1.3 Ερευνητικά και Τεχνικά Ερωτήματα 15

1.4 Πεδίο Εφαρμογής και Περιορισμοί 16

1.5 Δομή της Εργασίας 17

2. ΑΝΑΛΥΣΗ ΑΠΑΙΤΗΣΕΩΝ 18

2.1 Εισαγωγή στην Ανάλυση Απαιτήσεων 18

2.2 Λειτουργικές Απαιτήσεις 18

2.2.1 Παρουσίαση Καταλυμάτων 19

2.2.2 Ημερολόγιο Διαθεσιμότητας 20

2.2.3 Υποβολή Αιτήματος Κράτησης 22

2.2.4 Ροή Έγκρισης ή Απόρριψης Κράτησης 22

2.3 Μη Λειτουργικές Απαιτήσεις 23

2.4 Ρόλοι Χρηστών και Περιπτώσεις Χρήσης 23

2.5 Ανταγωνιστικά Συστήματα και Συγκριτική Αξιολόγηση 25

3. ΤΕΧΝΟΛΟΓΙΕΣ ΚΑΙ ΕΡΓΑΛΕΙΑ 29

3.1 Εισαγωγή στις Τεχνολογικές Επιλογές 29

3.2 Γλώσσα Προγραμματισμού Python 29
3.3 Πλαίσιο Ανάπτυξης(framework) Django 30
3.3.1 Γενική Περιγραφή του Django 30
3.3.2 Χρήση Django Apps και Modular Σχεδίαση 31
3.4 Βάση Δεδομένων και ORM 32
3.4.1 Χρήση Σχεσιακής Βάσης Δεδομένων 32
3.5 Διαχείριση Χρηστών και Ρόλων 33
3.6 Τεχνολογίες Διεπαφής Χρήστη 35
3.6.1 HTML Templates και Django Templating Engine 35
3.6.2 CSS και Οπτική Ταυτότητα 35
3.7 Εργαλεία Ανάπτυξης και Περιβάλλον Εργασίας 35
3.8 Συνολική Αξιολόγηση Τεχνολογικών Επιλογών 36
4. ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΣΥΣΤΗΜΑΤΟΣ 37
4.1 Γενική Αρχιτεκτονική Προσέγγιση 37
4.2 Δομή Εφαρμογής και Υποσυστήματα 37
4.2.1 Υποσύστημα Δημόσιας Διεπαφής 38
4.2.2 Υποσύστημα Κρατήσεων 38
4.2.3 Υποσύστημα Διαχείρισης (Admin Dashboard) 39
4.3 Ροές Δεδομένων και Επιχειρησιακή Λογική 40
4.4 Διαχείριση Συνέπειας και Συναλλαγών 40
5. ΣΧΕΔΙΑΣΗ ΒΑΣΗΣ ΔΕΔΟΜΕΝΩΝ 41
5.1 Εισαγωγή στη Σχεδίαση της Βάσης Δεδομένων 41
5.2 Ανάλυση Οντοτήτων και Συσχετίσεων 41
5.2.1 Οντότητα Καταλύματος 43
5.2.2 Οντότητα Κράτησης 43

5.2.3 Καταστάσεις και Κύκλος Ζωής Κράτησης 44
5.2.4 Οντότητα Περιόδου Μη Διαθεσιμότητας....45
5.2.5 Οντότητες Αιτημάτων Τροποποίησης και Ακύρωσης 46
5.2.6 Οντότητα Καταγραφής Ενεργειών 46
5.3 Κανόνες Ακεραιότητας και Επιχειρησιακοί Περιορισμοί....47
5.4 Διαχείριση Επικάλυψης Ημερομηνιών 47
6. ΥΛΟΠΟΙΗΣΗ ΣΥΣΤΗΜΑΤΟΣ 48
6.1 Εισαγωγή στην Υλοποίηση 48
6.2 Δρομολόγηση και Διαχείριση URLs 49
6.3 Υλοποίηση Δημόσιας Διεπαφής....49
6.4 Μηχανισμός Κρατήσεων και Διαθεσιμότητας 50
6.4.1 Έλεγχος Διαθεσιμότητας....51
6.4.2 Δημιουργία Αιτήματος Κράτησης....52
6.5 Διαχείριση Κρατήσεων από Ιδιοκτήτη....54
6.6 Ημερολόγιο και API Διαθεσιμότητας 54
6.7 Έλεγχος Πρόσβασης και Ρόλων Χρηστών....55
6.9 Συνολική Αξιολόγηση Υλοποίησης 56
7. ΔΙΕΠΑΦΗ ΧΡΗΣΤΗ ΚΑΙ ΕΜΠΕΙΡΙΑ ΧΡΗΣΤΗ (UI/UX)....57
7.1 Σχεδίαση και Αρχές UI/UX....57
7.2 Ροή Χρήστη για Υποβολή Κράτησης 59
7.3 Πίνακας Διαχείρισης Ιδιοκτήτη....62
8. ΕΛΕΓΧΟΣ ΚΑΙ ΔΟΚΙΜΕΣ....65
8.1 Στρατηγική Δοκιμών 65
8.2 Σενάρια και Περιπτώσεις Ελέγχου....66
8.3 Έλεγχοι Ασφάλειας και Εξουσιοδότησης 68

9. ΑΞΙΟΛΟΓΗΣΗ ΚΑΙ ΣΥΖΗΤΗΣΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ 70

9.1 Αντιστοίχιση Απαιτήσεων με Υλοποίηση 70

9.2 Περιορισμοί και Παραδοχές Συστήματος 72

10. ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ 75

10.1 Συμπεράσματα 75

10.2 Προτάσεις Μελλοντικής Εξέλιξης 76

ΒΙΒΛΙΟΓΡΑΦΙΑ 78

ΠΑΡΑΡΤΗΜΑ Α 79

Πίνακας Εικόνων

Εικόνα 1 Αρχική σελίδα: λίστα καταλυμάτων με τα βασικά χαρακτηριστικά τους και σύνοψη διαθεσιμότητας μήνα 20
Εικόνα 2 Προβολή λεπτομερειών καταλύματος 21
Εικόνα 3 Διάγραμμα περιπτώσεων χρήσης του συστήματος κρατήσεων 24
Πίνακας 1 Ποιοτική σύγκριση ανταγωνιστικών συστημάτων κρατήσεων με την εφαρμογή της παρούσας εργασίας 26
Πίνακας 2 Ποσοτική σύγκριση ετήσιου κόστους για ενδεικτικό κατάλυμα τριών μονάδων (493 διανυκτερεύσεις, κύκλος εργασιών ≈ 34.510 €) 27
Εικόνα 4 Οθόνη σύνδεσης χρήστη στο σύστημα 34
Εικόνα 5 Διάγραμμα κλάσεων των οντοτήτων της βάσης δεδομένων 42
Εικόνα 6 Διάγραμμα καταστάσεων του κύκλου ζωής μιας κράτησης 45
Εικόνα 7 Μενού - Λεπτομέρειες κράτησης, δυνατότητα υποβολής αιτήματος ακύρωσης, προβολής στοιχείων κράτησης και προβολή ιστορικού 47
Εικόνα 8 Διάγραμμα ακολουθίας για τη δημιουργία αιτήματος κράτησης 53
Εικόνα 9 Εμφάνιση στατιστικών στοιχείων στην αρχική οθόνη 58
Εικόνα 10 Έγκυρη επιλογή κράτησης 60
Εικόνα 11 Μη έγκυρη επιλογή κράτησης (Μη διαθέσιμες ημέρες απο άλλη κράτηση έχουν επιλεχθεί) 61
Εικόνα 12 Μενού - Οι κρατήσεις μου (Διαθέσιμο για συνδεδεμένους χρήστης) 62
Εικόνα 13 Μενού επιλογών διαχειριστή 63
Εικόνα 14 Προβολή και διαχείριση αιτημάτων μέσα απο την πλατφόρμα του διαχειριστή (Παράδειγμα - Επιβεβαίωση αίτηματος ακύρωσης κράτησης απο πελάτη) 64
Εικόνα 15 Επιλογή ημερομηνιών για κράτηση 79

ΚΕΦΑΛΑΙΟ 1ο

1. ΕΙΣΑΓΩΓΗ

1.1 Πλαίσιο και Αντικείμενο της Εργασίας

Η ραγδαία ανάπτυξη του διαδικτύου και των τεχνολογιών λογισμικού τα τελευταία χρόνια έχει μετασχηματίσει ουσιαστικά τον τρόπο με τον οποίο λειτουργεί ο τουριστικός κλάδος. Η ψηφιακή παρουσία των τουριστικών καταλυμάτων, σε συνδυασμό με αυτοματοποιημένα συστήματα διαχείρισης κρατήσεων, αποτελεί πλέον βασική προϋπόθεση για τη βιωσιμότητα και την ανταγωνιστικότητά τους. Οι σύγχρονοι χρήστες αναμένουν άμεση πληροφόρηση για τη διαθεσιμότητα, διαφανείς διαδικασίες κράτησης και ευχρηστία, ενώ οι ιδιοκτήτες και διαχειριστές καταλυμάτων απαιτούν εργαλεία που να εξασφαλίζουν ορθότητα δεδομένων, αποφυγή επικαλύψεων και αποτελεσματική διαχείριση αιτημάτων.

Στο πλαίσιο αυτό, το αντικείμενο της παρούσας εργασίας είναι η σχεδίαση και υλοποίηση ενός ολοκληρωμένου συστήματος κρατήσεων για τουριστικά καταλύματα, το οποίο λειτουργεί ως διαδικτυακή εφαρμογή και επιτρέπει την αλληλεπίδραση τόσο των επισκεπτών/πελατών όσο και των ιδιοκτητών ή διαχειριστών. Το σύστημα αναπτύχθηκε με τη χρήση του πλαισίου Django της γλώσσας Python, ακολουθώντας σύγχρονες αρχές ανάπτυξης διαδικτυακών εφαρμογών και διαχωρισμού ευθυνών (separation of concerns).

Η εφαρμογή επικεντρώνεται στη διαχείριση περιορισμένου αριθμού καταλυμάτων (δωματίων ή διαμερισμάτων), με στόχο την πληρότητα της λειτουργικότητας. Κεντρικό στοιχείο του συστήματος αποτελεί η μηχανή κρατήσεων, η οποία βασίζεται σε ημερολογιακή απεικόνιση της διαθεσιμότητας και υποστηρίζει πολλαπλές καταστάσεις (διαθέσιμο, μη διαθέσιμο, υπό αναμονή), επιτρέποντας την ασφαλή υποβολή αιτημάτων κράτησης.

1.2 Σκοπός και Στόχοι της Εργασίας

Ο κύριος σκοπός της εργασίας είναι η ανάπτυξη ενός λειτουργικού, αξιόπιστου και επεκτάσιμου συστήματος κρατήσεων πελατών για τουριστικά καταλύματα, το οποίο να καλύπτει πλήρως τις λειτουργικές και μη λειτουργικές απαιτήσεις που έχουν τεθεί. Η εργασία δεν περιορίζεται στη δημιουργία μιας απλής ιστοσελίδας παρουσίασης, αλλά επικεντρώνεται στον πυρήνα της επιχειρησιακής λογικής που διέπει τις κρατήσεις.

Οι βασικοί στόχοι της εργασίας συνοψίζονται ως εξής:

- Η υλοποίηση ενός μηχανισμού κρατήσεων που αποτρέπει πολλαπλές κρατήσεις στο ίδιο κατάλλυμα και για ίδιες ημερομηνίες και διασφαλίζει τη συνέπεια των δεδομένων.
- Η υποστήριξη ροής από αίτημα έως έγκριση, όπου ο πελάτης υποβάλλει αίτημα και ο ιδιοκτήτης αποφασίζει για την αποδοχή ή απόρριψή του.
- Η διάκριση ρόλων χρηστών (επισκέπτης, εγγεγραμμένος πελάτης, ιδιοκτήτης/διαχειριστής) με διαφορετικά δικαιώματα πρόσβασης για τον κάθε ρόλο.
- Η παροχή εργαλείων διαχείρισης στον ιδιοκτήτη και διαχείριση αιτημάτων τροποποίησης ή ακύρωσης.
- Η αξιοποίηση σύγχρονων τεχνολογιών backend με έμφαση στην καθαρή αρχιτεκτονική και την επεκτασιμότητα.

Μέσα από τους παραπάνω στόχους, επιδιώκεται να αναδειχθούν τόσο οι τεχνικές πτυχές της ανάπτυξης λογισμικού όσο και τη συσχέτισή τους με πραγματικές επιχειρησιακές ανάγκες στον τουριστικό τομέα.

1.3 Ερευνητικά και Τεχνικά Ερωτήματα

Η υλοποίηση ενός συστήματος κρατήσεων αναδεικνύει μια σειρά ερευνητικών και τεχνικών ερωτημάτων, τα οποία αποτέλεσαν καθοδηγητικό άξονα κατά τον σχεδιασμό και την ανάπτυξη της εφαρμογής. Ενδεικτικά, τα βασικότερα ερωτήματα είναι τα εξής:

- Πώς μπορεί να σχεδιαστεί ένας μηχανισμός διαθεσιμότητας που να λαμβάνει υπόψη τόσο εγκεκριμένες κρατήσεις όσο και προσωρινά αιτήματα (αναμονές που δεν είναι ακόμα επιβεβαιωμένες);
- Ποιος είναι ο καταλληλότερος τρόπος μοντελοποίησης των καταστάσεων μιας κράτησης (εκκρεμής, εγκεκριμένη, απορριφθείσα, ακυρωμένη) ώστε να αποφεύγονται ασυνέπειες;
- Με ποιον τρόπο μπορεί να διασφαλιστεί η ακεραιότητα των δεδομένων σε περιβάλλον ταυτόχρονων αιτημάτων;
- Πώς υλοποιείται αποτελεσματικά ο διαχωρισμός ρόλων και δικαιωμάτων πρόσβασης σε μια διαδικτυακή εφαρμογή;
- Πώς μπορεί μια εφαρμογή Django να οργανωθεί σε υποσυστήματα ώστε να παραμένει κατανοητή και επεκτάσιμη;

1.4 Πεδίο Εφαρμογής και Περιορισμοί

Το πεδίο εφαρμογής της παρούσας εργασίας περιορίζεται στη δημιουργία ενός αυτοτελούς συστήματος κρατήσεων για μικρής κλίμακας τουριστικά καταλύματα. Η επιλογή αυτή έγινε σκόπιμα, ώστε η έμφαση να δοθεί στη σωστή υλοποίηση της επιχειρησιακής λογικής και όχι στην πολυπλοκότητα μεγάλης κλίμακας συστημάτων.

Σημαντικοί περιορισμοί της εργασίας περιλαμβάνουν:

- Η μη ενσωμάτωση συστημάτων ηλεκτρονικών πληρωμών ή τραπεζικών υπηρεσιών.
- Η απουσία εξωτερικών καναλιών διασύνδεσης (π.χ. πλατφόρμες τύπου OTA).
- Η εστίαση στο backend και στη λογική κρατήσεων, με δευτερεύοντα ρόλο στον αισθητικό σχεδιασμό της διεπαφής.
- Η υλοποίηση σε περιβάλλον ανάπτυξης, χωρίς απαιτήσεις κλιμάκωσης ή κατανεμημένης αρχιτεκτονικής.

Παρά τους περιορισμούς αυτούς, το σύστημα έχει σχεδιαστεί με τρόπο που επιτρέπει μελλοντικές επεκτάσεις, τόσο λειτουργικές όσο και τεχνολογικές.

1.5 Δομή της Εργασίας

Η παρούσα εργασία οργανώνεται σε δέκα κεφάλαια, τα οποία καλύπτουν σταδιακά όλες τις πτυχές της ανάπτυξης του συστήματος. Μετά την εισαγωγή, το Κεφάλαιο 2 παρουσιάζει την ανάλυση απαιτήσεων, διαχωρίζοντας τις λειτουργικές από τις μη λειτουργικές απαιτήσεις. Το Κεφάλαιο 3 αναλύει τις τεχνολογίες και τα εργαλεία που επιλέχθηκαν, ενώ το Κεφάλαιο 4 εστιάζει στην αρχιτεκτονική του συστήματος.

Ακολουθούν κεφάλαια που αφορούν τον σχεδιασμό της βάσης δεδομένων, την υλοποίηση του συστήματος, τη διεπαφή χρήστη και τις διαδικασίες ελέγχου. Τέλος, τα τελευταία κεφάλαια παρουσιάζουν την αξιολόγηση του έργου, τα συμπεράσματα και τις προτάσεις για μελλοντική εξέλιξη.

Με τη δομή αυτή, η εργασία επιδιώκει να προσφέρει μια ολοκληρωμένη και τεκμηριωμένη παρουσίαση της ανάπτυξης ενός σύγχρονου συστήματος κρατήσεων, συνδυάζοντας λεπτομερή ανάλυση και υλοποίηση.

ΚΕΦΑΛΑΙΟ 2ο

2. ΑΝΑΛΥΣΗ ΑΠΑΙΤΗΣΕΩΝ

2.1 Εισαγωγή στην Ανάλυση Απαιτήσεων

Η ανάλυση απαιτήσεων αποτελεί θεμελιώδες στάδιο στον κύκλο ζωής της ανάπτυξης λογισμικού, καθώς καθορίζει το λειτουργικό και ποιοτικό πλαίσιο εντός του οποίου θα σχεδιαστεί και θα υλοποιηθεί το σύστημα. Στην παρούσα εργασία, η ανάλυση απαιτήσεων λειτουργεί ως εννοιολογική και τεχνική γέφυρα μεταξύ των αναγκών ενός πραγματικού σεναρίου τουριστικών κρατήσεων και της τελικής υλοποίησης μιας διαδικτυακής εφαρμογής βασισμένης στο Django. Η ορθότητα και πληρότητα της ανάλυσης αυτής επηρεάζει άμεσα τόσο τη λειτουργική επάρκεια του συστήματος όσο και τη μελλοντική του επεκτασιμότητα (Sommerville, 2016).

Η ανάλυση οργανώνεται σε λειτουργικές και μη λειτουργικές απαιτήσεις, ακολουθούμενη από παρουσίαση των ρόλων χρηστών, των περιπτώσεων χρήσης και των κριτηρίων αποδοχής. Η δομή αυτή επιλέχθηκε ώστε να καλύπτει τόσο το «τι» κάνει το σύστημα όσο και το «πώς» και «υπό ποιες συνθήκες» οφείλει να το κάνει.

2.2 Λειτουργικές Απαιτήσεις

Οι λειτουργικές απαιτήσεις περιγράφουν τις βασικές λειτουργίες και υπηρεσίες που οφείλει να παρέχει το σύστημα στους χρήστες του. Στο πλαίσιο της εφαρμογής κρατήσεων που αναπτύχθηκε, οι απαιτήσεις αυτές επικεντρώνονται στη διαχείριση καταλυμάτων, στον ακριβή υπολογισμό της διαθεσιμότητας, στη διαδικασία υποβολής αιτημάτων κράτησης και στη ροή διαχειριστικής απόφασης από τον ιδιοκτήτη. Οι λειτουργικές απαιτήσεις δεν αντιμετωπίζονται

ως ανεξάρτητα χαρακτηριστικά, αλλά ως αλληλένδετα στάδια ενός ενιαίου κύκλου ζωής κράτησης.

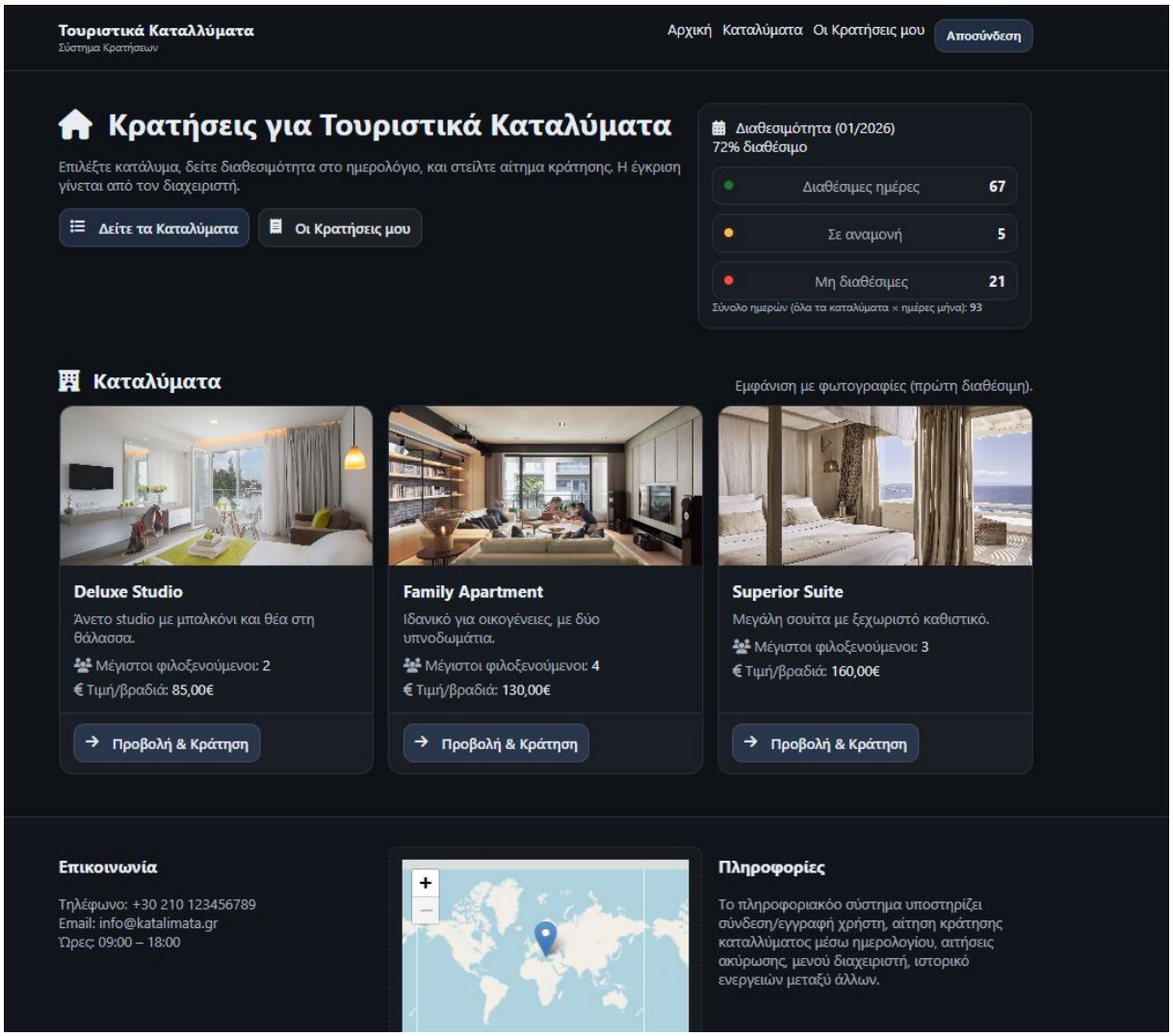
Στον πηγαίο κώδικα, η λογική αυτή αποτυπώνεται με τη συγκέντρωση κρίσιμων επιχειρησιακών κανόνων σε κεντρικές υπηρεσίες (services), όπως το reservations/services.py, οι οποίες καλούνται από διαφορετικά σημεία της εφαρμογής. Έτσι, η ίδια απαίτηση (π.χ. έλεγχος διαθεσιμότητας) ικανοποιείται με ενιαίο τρόπο, ανεξάρτητα από το αν η ενέργεια προέρχεται από δημόσιο χρήστη, από εγγεγραμμένο πελάτη ή από τον ιδιοκτήτη μέσω του dashboard.

2.2.1 Παρουσίαση Καταλυμάτων

Μία από τις βασικότερες λειτουργικές απαιτήσεις του συστήματος είναι η δυνατότητα δημόσιας παρουσίασης των διαθέσιμων καταλυμάτων. Η απαίτηση αυτή εξυπηρετεί την ανάγκη του επισκέπτη να αποκτήσει σαφή και άμεση εικόνα των επιλογών διαμονής, χωρίς να απαιτείται προηγούμενη εγγραφή ή σύνδεση στο σύστημα. Στην εφαρμογή, κάθε κατάλυμα περιγράφεται από βασικά χαρακτηριστικά όπως ονομασία, αναλυτική περιγραφή, μέγιστος αριθμός φιλοξενούμενων και βασική τιμή ανά διανυκτέρευση, στοιχεία που αντιστοιχούν στα πεδία του μοντέλου Accommodation.

Η παρουσίαση των καταλυμάτων υλοποιείται μέσω της αρχικής σελίδας και των επιμέρους σελίδων λεπτομερειών. Στο επίπεδο του backend, τα δεδομένα ανακτώνται δυναμικά από τη βάση δεδομένων και αποδίδονται στα templates, ενώ στο frontend οργανώνονται με τρόπο που διευκολύνει τη σύγκριση και την κατανόηση Η απαίτηση αυτή δεν περιορίζεται στη στατική προβολή πληροφοριών. Η παρουσίαση των καταλυμάτων συνδέεται άμεσα με τον μηχανισμό διαθεσιμότητας, καθώς ο επισκέπτης πρέπει να μπορεί να μεταβεί ομαλά από τη γενική εικόνα ενός καταλύματος στην εξέταση της διαθεσιμότητάς του. Η σύνδεση αυτή αποτελεί προϋπόθεση για τις επόμενες λειτουργικές απαιτήσεις και επηρεάζει τη συνολική συνοχή της εφαρμογής.

Η δημόσια αυτή προβολή παρουσιάζεται στην Εικόνα 1, όπου κάθε κατάλυμα εμφανίζεται με την ονομασία, τη σύντομη περιγραφή, τον μέγιστο αριθμό φιλοξενούμενων και τη βασική τιμή ανά διανυκτέρευση.



Εικόνα 1 Αρχική σελίδα: λίστα καταλυμάτων με τα βασικά χαρακτηριστικά τους και σύνοψη διαθεσιμότητας μήνα

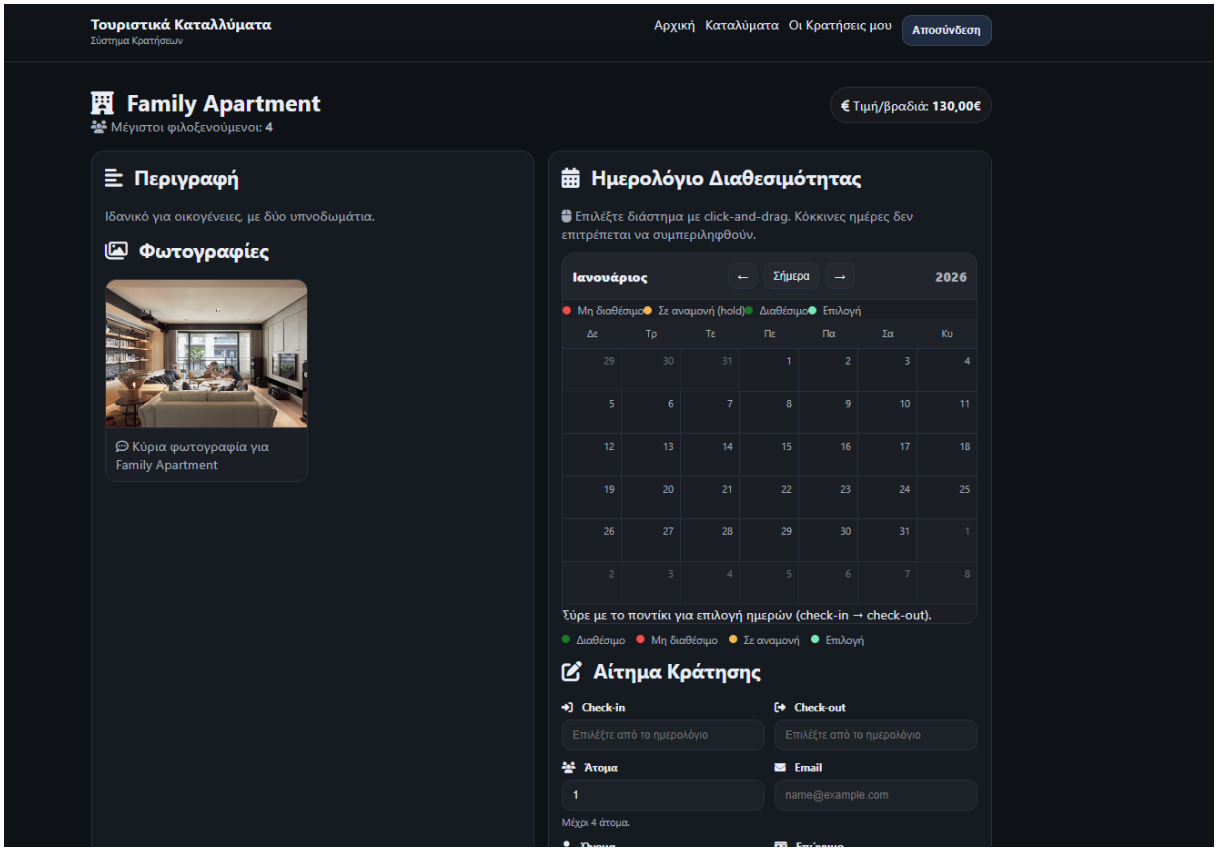
2.2.2 Ημερολόγιο Διαθεσιμότητας

Κεντρική λειτουργική απαίτηση του συστήματος αποτελεί η απεικόνιση της διαθεσιμότητας των καταλυμάτων σε μορφή ημερολογίου. Το ημερολόγιο δεν λειτουργεί απλώς ως οπτικό βοήθημα, αλλά ως ακριβής αναπαράσταση της επιχειρησιακής κατάστασης του συστήματος σε συγκεκριμένες χρονικές περιόδους. Η ακρίβεια της πληροφορίας αυτής είναι μεγάλης σημασίας, καθώς οποιαδήποτε ασυνέπεια μπορεί να οδηγήσει σε αντικρουόμενες κρατήσεις ή σε απώλεια εμπιστοσύνης από τους χρήστες.

Η διαθεσιμότητα υπολογίζεται δυναμικά με βάση τρεις βασικούς παράγοντες:

- τις εγκεκριμένες κρατήσεις,
- τις περιόδους μη διαθεσιμότητας (blocked periods)
- και τα ενεργά εκκρεμή αιτήματα κράτησης που δεσμεύουν προσωρινά ημερομηνίες μέσω του μηχανισμού κρατήσεων.

Η Εικόνα 2 παρουσιάζει το ημερολόγιο διαθεσιμότητας ενός καταλύματος. Οι τρεις καταστάσεις μιας ημέρας, δηλαδή διαθέσιμη, σε αναμονή λόγω ενεργής προσωρινής δέσμευσης και μη διαθέσιμη, διακρίνονται με ξεχωριστό χρωματικό κωδικό, ενώ η επιλογή διαστήματος γίνεται απευθείας πάνω στο ημερολόγιο.



Εικόνα 2 Προβολή λεπτομερειών καταλύματος

2.2.3 Υποβολή Αιτήματος Κράτησης

Το σύστημα οφείλει να επιτρέπει στον επισκέπτη ή πελάτη να υποβάλει αίτημα κράτησης για συγκεκριμένο κατάλυμα και χρονικό διάστημα. Η λειτουργική αυτή απαίτηση υλοποιείται μέσω φόρμας υποβολής, στην οποία συλλέγονται βασικά προσωπικά στοιχεία καθώς και πληροφορίες σχετικές με τη διαμονή. Κατά την υποβολή αιτήματος, το σύστημα εκτελεί διαδοχικούς ελέγχους εγκυρότητας. Αρχικά, επαληθεύεται ότι οι ημερομηνίες είναι έγκυρες και ότι το διάστημα check-in/check-out έχει σωστή χρονική σειρά. Στη συνέχεια, ελέγχεται ότι ο αριθμός φιλοξενούμενων δεν υπερβαίνει το μέγιστο όριο του καταλύματος. Τέλος, πραγματοποιείται έλεγχος διαθεσιμότητας για το επιλεγμένο διάστημα, πριν δημιουργηθεί εγγραφή κράτησης σε κατάσταση «ΣΕ ΕΚΚΡΕΜΟΤΗΤΑ».

Ιδιαίτερης σημασίας είναι ο μηχανισμός προσωρινής δέσμευσης ημερομηνιών. Με την εισαγωγή ενός χρονικού ορίου, το σύστημα αποτρέπει τη δημιουργία πολλαπλών ανταγωνιστικών αιτημάτων για το ίδιο διάστημα, χωρίς όμως να προχωρά σε οριστική κράτηση πριν την απόφαση του ιδιοκτήτη.

2.2.4 Ροή Έγκρισης ή Απόρριψης Κράτησης

Η διαδικασία κράτησης δεν ολοκληρώνεται αυτόματα, αλλά απαιτεί διαχειριστική απόφαση από τον ιδιοκτήτη ή διαχειριστή των καταλυμάτων. Η λειτουργική αυτή απαίτηση υλοποιείται μέσω ειδικού dashboard, στο οποίο εμφανίζονται όλα τα εκκρεμή αιτήματα. Ο ιδιοκτήτης έχει τη δυνατότητα να εξετάζει τα στοιχεία κάθε αιτήματος και να λαμβάνει απόφαση έγκρισης ή απόρριψης. Κατά την έγκριση μιας κράτησης, το σύστημα επαναλαμβάνει τον έλεγχο διαθεσιμότητας, διασφαλίζοντας ότι δεν έχει μεσολαβήσει άλλη δέσμευση στο ίδιο χρονικό διάστημα. Η κράτηση μεταβαίνει σε κατάσταση «ΕΠΙΒΕΒΑΙΩΜΕΝΗ» και οι ημερομηνίες δεσμεύονται οριστικά. Αντίθετα, σε περίπτωση απόρριψης, η κράτηση μεταβαίνει σε κατάσταση «ΑΠΟΡΡΙΦΘΕΙΣΑ» και οι ημερομηνίες αποδεσμεύονται, καθιστώντας τις ξανά διαθέσιμες. Η ροή αυτή υλοποιείται σε επίπεδο υπηρεσιών με χρήση συναλλαγών (transaction.atomic), εξασφαλίζοντας ακεραιότητα ακόμη και σε σενάρια ταυτόχρονης πρόσβασης.

2.3 Μη Λειτουργικές Απαιτήσεις

Πέρα από τις λειτουργικές δυνατότητες, το σύστημα πρέπει να ικανοποιεί και μη λειτουργικές απαιτήσεις που αφορούν την ποιότητα, την ασφάλεια και την ευχρηστία. Οι απαιτήσεις αυτές δεν είναι άμεσα ορατές στον τελικό χρήστη, αλλά επηρεάζουν σημαντικά την αξιοπιστία και τη βιωσιμότητα της εφαρμογής (ISO/IEC, 2011).

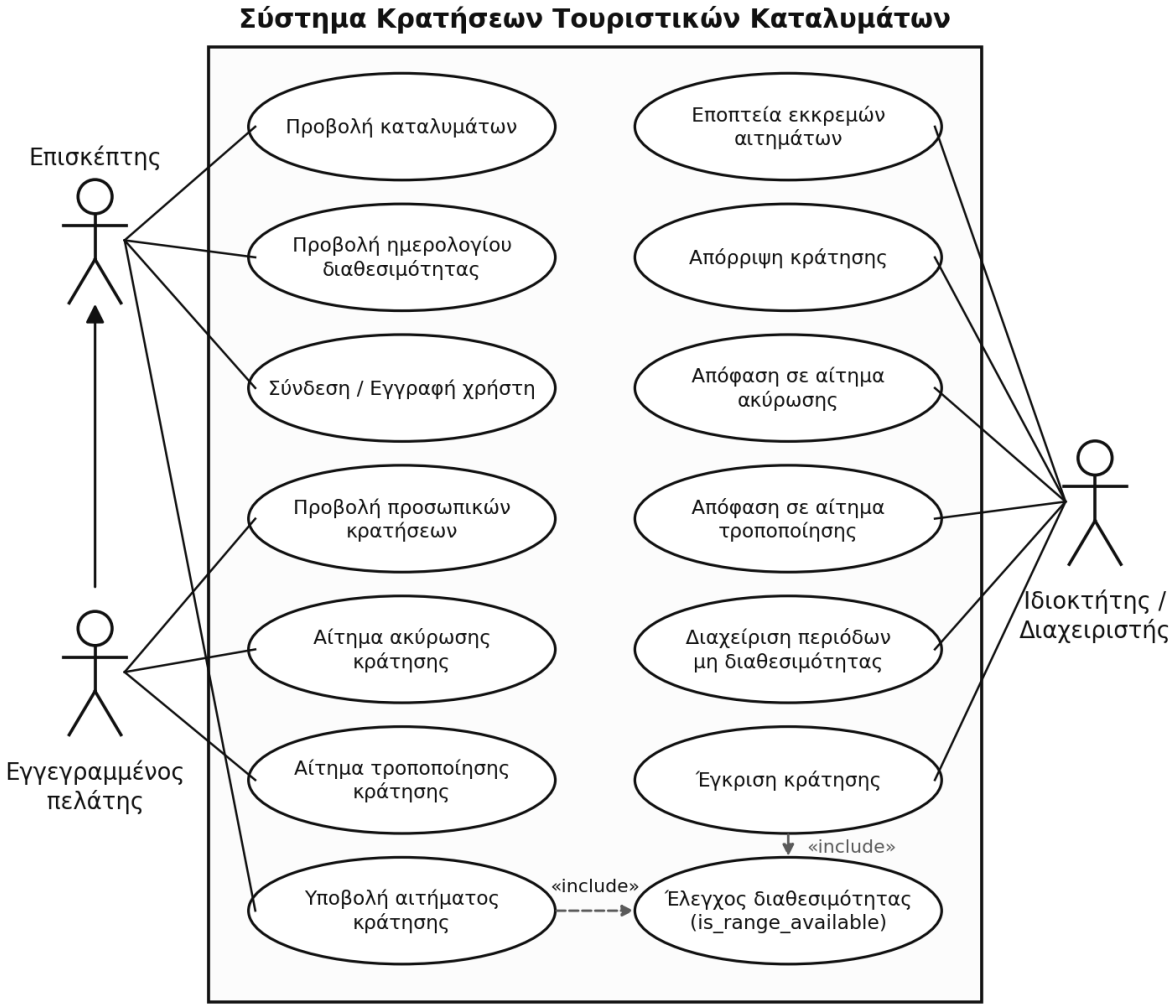
- **Αξιοπιστία και Ακεραιότητα Δεδομένων:** Η αξιοπιστία του συστήματος βασίζεται στη διασφάλιση ότι δεν μπορούν να δημιουργηθούν αντικρουόμενες κρατήσεις και ότι οι καταστάσεις των κρατήσεων παραμένουν συνεπείς. Η χρήση κεντρικών ελέγχων διαθεσιμότητας, συναλλαγών και μηχανισμών καταγραφής ενεργειών (logs) συμβάλλει στη διατήρηση της ακεραιότητας των δεδομένων.
- **Ασφάλεια και Έλεγχος Πρόσβασης:** Η ασφάλεια υλοποιείται μέσω διακριτών ρόλων χρηστών και ελέγχων πρόσβασης σε επίπεδο views και queries. Μόνο εξουσιοδοτημένοι χρήστες έχουν πρόσβαση στο dashboard διαχείρισης, ενώ οι απλοί χρήστες περιορίζονται στις δημόσιες λειτουργίες και στις δικές τους κρατήσεις (OWASP Foundation, 2023a).
- **Ευχρηστία:** Η ευχρηστία του συστήματος επιτυγχάνεται μέσω απλής και σαφούς διεπαφής, δυναμικού ημερολογίου διαθεσιμότητας με εύχρηστο και απλό τρόπο για κράτηση των ημερομηνιών και ξεκάθαρων μηνυμάτων επιβεβαίωσης ή σφάλματος (Nielsen, 2024).

2.4 Ρόλοι Χρηστών και Περιπτώσεις Χρήσης

Το σύστημα υποστηρίζει διακριτούς ρόλους χρηστών, οι οποίοι αλληλεπιδρούν με διαφορετικό τρόπο με την εφαρμογή. Ο επισκέπτης έχει πρόσβαση στην προβολή καταλυμάτων και στη δυνατότητα υποβολής αιτήματος κράτησης, ο εγγεγραμμένος πελάτης μπορεί να παρακολουθεί τις κρατήσεις του και να υποβάλλει αιτήματα αλλαγών, ενώ ο ιδιοκτήτης διαχειρίζεται το σύνολο των αιτημάτων και της διαθεσιμότητας.

Οι περιπτώσεις χρήσης προκύπτουν φυσικά από τους ρόλους αυτούς και καλύπτουν ολόκληρο τον κύκλο ζωής μιας κράτησης, από την αρχική αναζήτηση έως την τελική απόφαση και την ενδεχόμενη ακύρωση ή τροποποίηση.

Οι ρόλοι και οι περιπτώσεις χρήσης που περιγράφηκαν παραπάνω αποτυπώνονται συνοπτικά στο διάγραμμα περιπτώσεων χρήσης (use case diagram) της Εικόνας 3, το οποίο ακολουθεί τη σημειογραφία UML 2.5.1 (Object Management Group, 2017). Στο διάγραμμα διακρίνονται οι τρεις ρόλοι, οι λειτουργίες που αντιστοιχούν σε καθέναν, η σχέση γενίκευσης μεταξύ επισκέπτη και εγγεγραμμένου πελάτη, καθώς και η σχέση «include» με τον κεντρικό έλεγχο διαθεσιμότητας, ο οποίος επαναχρησιμοποιείται τόσο κατά την υποβολή αιτήματος όσο και κατά την έγκρισή του.



Εικόνα 3 Διάγραμμα περιπτώσεων χρήσης του συστήματος κρατήσεων

2.5 Ανταγωνιστικά Συστήματα και Συγκριτική Αξιολόγηση

Η διατύπωση των απαιτήσεων ενός συστήματος κρατήσεων δεν μπορεί να γίνει ανεξάρτητα από το υφιστάμενο τεχνολογικό περιβάλλον, καθώς ένα τουριστικό κατάλυμα μικρής κλίμακας έχει ήδη στη διάθεσή του πλήθος έτοιμων λύσεων. Η παρούσα ενότητα εξετάζει τα βασικά ανταγωνιστικά συστήματα, καταγράφει τα χαρακτηριστικά τους και τα συγκρίνει τόσο ποιοτικά όσο και ποσοτικά με την εφαρμογή που αναπτύχθηκε. Στόχος είναι ο σαφής προσδιορισμός της θέσης που καταλαμβάνει η παρούσα εργασία στο πεδίο και η τεκμηρίωση των απαιτήσεων που τέθηκαν στα προηγούμενα υποκεφάλαια.

Τα υπάρχοντα συστήματα μπορούν να ταξινομηθούν σε τρεις κατηγορίες. Η πρώτη περιλαμβάνει τις πλατφόρμες **διαδικτυακής διάθεσης** (Online Travel Agencies, OTA), όπως η Booking.com και η Airbnb, οι οποίες λειτουργούν ως κανάλια διάθεσης και όχι ως λογισμικό εγκατεστημένο στο κατάλυμα. Η δεύτερη περιλαμβάνει **εμπορικά συστήματα διαχείρισης** καταλυμάτων που προσφέρονται ως υπηρεσία (SaaS) και συνδυάζουν σύστημα διαχείρισης, μηχανή κρατήσεων και διαχειριστή καναλιών, με χαρακτηριστικό παράδειγμα το Smoobu. Η τρίτη περιλαμβάνει **λύσεις ελεύθερου λογισμικού και ανοικτού κώδικα**, όπως το HotelDruid και το QloApps, οι οποίες μπορούν να εγκατασταθούν σε υποδομή του ίδιου του καταλύματος.

Η Booking.com λειτουργεί με μοντέλο προμήθειας, δηλαδή το κατάλυμα καταβάλλει ένα ποσοστό επί της αξίας κάθε κράτησης που πραγματοποιείται μέσω της πλατφόρμας, το ποσοστό διαφοροποιείται ανά χώρα και τύπο καταλύματος, με μέσο όρο περίπου 15%, και χρεώνεται μετά την ολοκλήρωση της διαμονής (Booking.com). Το σημαντικότερο χαρακτηριστικό από τεχνική άποψη είναι ότι η επιβεβαίωση της κράτησης είναι άμεση και η «πηγή αλήθειας» για τη διαθεσιμότητα βρίσκεται στην πλατφόρμα και όχι στο κατάλυμα.

Η Airbnb παρουσιάζει ιδιαίτερο ενδιαφέρον για την παρούσα εργασία, διότι υποστηρίζει και τις δύο βασικές ροές κράτησης. Στη λειτουργία άμεσης κράτησης (Instant Book) ο επισκέπτης δεσμεύει απευθείας τις ημερομηνίες, ενώ στη λειτουργία αιτήματος κράτησης ο οικοδεσπότης έχει περιθώριο 24 ωρών να αποδεχθεί ή να απορρίψει το αίτημα, εάν δεν απαντήσει, το αίτημα λήγει και οι ημερομηνίες αποδεσμεύονται (Airbnb). Η ροή αυτή είναι εννοιολογικά ισοδύναμη με τον μηχανισμό προσωρινής δέσμευσης που υλοποιείται στην παρούσα εργασία μέσω του πεδίου hold_expires_at και της ρύθμισης BOOKING_HOLD_MINUTES.

Το Smoobu αποτελεί χαρακτηριστικό εμπορικό σύστημα SaaS που απευθύνεται σε διαχειριστές λίγων μονάδων και συνδυάζει σύστημα διαχείρισης, δημιουργό ιστοσελίδας με ενσωματωμένη μηχανή κρατήσεων και διαχειριστή καναλιών προς περισσότερες από εκατό πλατφόρμες. Η χρέωση είναι συνδρομητική ανά μονάδα και ανά μήνα, χωρίς προμήθεια επί των κρατήσεων στο ετήσιο πρόγραμμα (Smoobu).

Το HotelDruid διατίθεται με την άδεια ελεύθερου λογισμικού AGPL και καλύπτει καταλύματα από λίγα διαμερίσματα έως ξενοδοχεία εκατοντάδων δωματίων, προσφέροντας ημερολόγιο, πολυχρηστική λειτουργία με δικαιώματα και αναφορές, η μηχανή κρατήσεων και ο διαχειριστής καναλιών διατίθενται ως πρόσθετες, εμπορικές μονάδες (HotelDruid). Αντίστοιχα, το QloApps αποτελεί ανοικτού κώδικα σύστημα διαχείρισης ξενοδοχείων με ενσωματωμένη μηχανή κρατήσεων και διαχειριστή καναλιών, με το ίδιο το λογισμικό να διατίθεται δωρεάν και τη χρέωση να αφορά τη φιλοξενία και την υποστήριξη (QloApps).

Στον Πίνακα 1 συνοψίζεται η ποιοτική σύγκριση των παραπάνω συστημάτων με την εφαρμογή που αναπτύχθηκε, ως προς την κατηγορία, το μοντέλο κόστους, τη διαθεσιμότητα του πηγαίου κώδικα, την κύρια ροή κράτησης και τη δυνατότητα διαχείρισης καναλιών.

Πίνακας 1 Ποιοτική σύγκριση ανταγωνιστικών συστημάτων κρατήσεων με την εφαρμογή της παρούσας εργασίας

Σύστημα	**Κατηγορία**	**Μοντέλο κόστους**	**Πηγαίος κώδικας / εγκατάσταση**	**Κύρια ροή κράτησης**	**Διαχείριση καναλιών (OTA)**
Booking.com	Πλατφόρμα διάθεσης (OTA)	Προμήθεια επί της αξίας κάθε κράτησης (μέσος όρος ~ 15%, διαφοροποιείται ανά χώρα και τύπο καταλύματος)	Κλειστός, φιλοξενούμενη υπηρεσία	Άμεση επιβεβαίωση κράτησης από την πλατφόρμα	Αποτελεί η ίδια κανάλι διάθεσης
Airbnb	Πλατφόρμα διάθεσης (OTA)	Προμήθεια ανά κράτηση	Κλειστός, φιλοξενούμενη υπηρεσία	Instant Book ή αίτημα κράτησης με απόκριση οικοδεσπότη εντός 24 ωρών	Αποτελεί η ίδια κανάλι διάθεσης
Smoobu	Εμπορικό SaaS (διαχείριση, μηχανή κρατήσεων, κανάλια)	Συνδρομή ανά μονάδα και μήνα, 0% προμήθεια στο ετήσιο πρόγραμμα	Κλειστός, υπηρεσία cloud	Άμεση κράτηση μέσω μηχανής κρατήσεων στον ιστότοπο του καταλύματος	Ναι, προς 100+ κανάλια
HotelDruid	Ελεύθερο λογισμικό / ανοικτός κώδικας (AGPL)	Δωρεάν βασικές λειτουργίες, χρέωση για φιλοξενία και πρόσθετες μονάδες	Ανοικτός, δυνατότητα αυτοφιλοξενίας	Κρατήσεις μέσω προαιρετικής μονάδας μηχανής κρατήσεων	Προαιρετική μονάδα (Booking.com, Expedia)
QloApps	Ελεύθερο λογισμικό /	Δωρεάν λογισμικό, χρέωση για φιλοξενία και	Ανοικτός, δυνατότητα	Άμεση κράτηση μέσω ενσωματωμένης μηχανής	Ναι, ενσωματωμένος

	ανοικτός κώδικας	υποστήριξη	αυτοφιλοξενίας	κρατήσεων	διαχειριστής καναλιών
Παρούσα εργασία	Ακαδημαϊκό πρωτότυπο (Django)	Χωρίς κόστος αδειοδότησης, μόνο κόστος φιλοξενίας	Πλήρης πρόσβαση στον κώδικα, αυτοφιλοξενία	Αίτημα κράτησης με ρητή προσωρινή δέσμευση και απόφαση ιδιοκτήτη	Δεν υποστηρίζεται (εκτός πεδίου εφαρμογής)

Από την ποιοτική σύγκριση προκύπτει ότι η βασική διαφοροποίηση δεν εντοπίζεται στο πλήθος των λειτουργιών, αλλά στο πού βρίσκεται η «πηγή αλήθειας» για τη διαθεσιμότητα και στο ποιος ελέγχει την απόφαση της κράτησης. Οι πλατφόρμες διάθεσης μεταφέρουν και τα δύο εκτός του καταλύματος, τα εμπορικά συστήματα SaaS τα διατηρούν σε υποδομή τρίτου παρόχου, ενώ οι λύσεις ανοικτού κώδικα και η παρούσα εργασία τα διατηρούν στο ίδιο το κατάλυμα. Η επιλογή αυτή εξηγεί και τη διαφορά στο μοντέλο κόστους, το οποίο εξετάζεται ποσοτικά στη συνέχεια.

Για τη σύγκριση υιοθετείται ένα ενδεικτικό σενάριο μικρού καταλύματος με τρεις μονάδες, μέση τιμή 70 € ανά διανυκτέρευση και ετήσια πληρότητα 45%. Το σενάριο αυτό αντιστοιχεί σε 493 διανυκτερεύσεις και ετήσιο κύκλο εργασιών περίπου 34.510 €. Τα ποσοστά προμήθειας και οι τιμές συνδρομής προέρχονται από την τεκμηρίωση των αντίστοιχων παρόχων, ενώ το κόστος φιλοξενίας για τις αυτοφιλοξενούμενες λύσεις αποτελεί παραδοχή και όχι τεκμηριωμένη τιμή. Τα αποτελέσματα παρουσιάζονται στον Πίνακα 2.

Πίνακας 2 Ποσοτική σύγκριση ετήσιου κόστους για ενδεικτικό κατάλυμα τριών μονάδων (493 διανυκτερεύσεις, κύκλος εργασιών ≈ 34.510 €)

Λύση	**Ετήσιο κόστος λογισμικού**	**Προμήθεια ανά κράτηση**	**Εκτιμώμενο ετήσιο κόστος**
Booking.com (100% των κρατήσεων μέσω OTA)	—	~ 15%	~ 5.177 €
Booking.com (50% των κρατήσεων μέσω OTA)	—	~ 15%	~ 2.588 €
Smoobu (Professional Prepaid, 3 μονάδες)	608,40 € (31,50 € + 2 × 9,60 € ανά μήνα)	0%	~ 608 €
HotelDruid / QloApps (αυτοφιλοξενία)	0 € (άδεια χρήσης)	0%	Κόστος φιλοξενίας (60-120 €)
Παρούσα εργασία (αυτοφιλοξενία)	0 €	0%	Κόστος φιλοξενίας (60-120 €)

Η ποσοτική σύγκριση αναδεικνύει ότι το μοντέλο προμήθειας γίνεται ασύμφορο καθώς αυξάνεται ο κύκλος εργασιών, ενώ το συνδρομητικό και το αυτοφιλοξενούμενο μοντέλο έχουν σταθερό κόστος. Ταυτόχρονα, όμως, η σύγκριση δεν είναι μονοδιάστατη αφού οι πλατφόρμες διάθεσης προσφέρουν πρόσβαση σε κοινό που ένα αυτόνομο σύστημα δεν διαθέτει, ενώ τα εμπορικά συστήματα SaaS καλύπτουν τη συντήρηση και τη διασύνδεση με κανάλια, κόστος που στην περίπτωση της αυτοφιλοξενίας μετατοπίζεται στον ίδιο τον διαχειριστή.

Η παρούσα εργασία δεν ανταγωνίζεται τα παραπάνω συστήματα ως προς την εμπορική πληρότητα, καθώς δεν ενσωματώνει πληρωμές, δυναμική τιμολόγηση ή διασύνδεση με κανάλια διάθεσης. Η συνεισφορά της εντοπίζεται στο ότι υλοποιεί ρητά και τεκμηριωμένα τον πυρήνα που τα εμπορικά συστήματα αντιμετωπίζουν ως κλειστό «μαύρο κουτί», δηλαδή την ενιαία συνάρτηση ελέγχου διαθεσιμότητας, τη ρητή προσωρινή δέσμευση με χρονικό όριο, τη ροή απόφασης εντός συναλλαγής με κλείδωμα εγγραφών και την πλήρη καταγραφή ενεργειών. Ως εκ τούτου, τοποθετείται μεταξύ των λύσεων ανοικτού κώδικα και των ακαδημαϊκών πρωτοτύπων, με έμφαση στην ορθότητα της επιχειρησιακής λογικής και στη δυνατότητα ελέγχου της.

ΚΕΦΑΛΑΙΟ 3ο

3. ΤΕΧΝΟΛΟΓΙΕΣ ΚΑΙ ΕΡΓΑΛΕΙΑ

3.1 Εισαγωγή στις Τεχνολογικές Επιλογές

Η επιλογή των κατάλληλων τεχνολογιών αποτελεί βασικό παράγοντα για την επιτυχία ενός συστήματος λογισμικού, ιδιαίτερα όταν αυτό αφορά εφαρμογές διαδικτύου με σύνθετη επιχειρησιακή λογική, όπως τα συστήματα κρατήσεων. Στην παρούσα εργασία, οι τεχνολογικές επιλογές έγιναν με γνώμονα την αξιοπιστία, τη σαφή δομή, την επεκτασιμότητα και τη δυνατότητα καθαρής μοντελοποίησης των απαιτήσεων. Η εφαρμογή υλοποιήθηκε ως server-side web application, βασισμένη σε αρχιτεκτονική Model-View-Template (MVT), χρησιμοποιώντας το Django framework της γλώσσας Python. Η επιλογή αυτή επέτρεψε τη συγκέντρωση της επιχειρησιακής λογικής στο backend, για τη σωστή διαχείριση κρατήσεων, ημερομηνιών και καταστάσεων.

Στις επόμενες ενότητες παρουσιάζονται αναλυτικά οι τεχνολογίες και τα εργαλεία που χρησιμοποιήθηκαν, καθώς και ο ρόλος τους στο συνολικό σύστημα.

3.2 Γλώσσα Προγραμματισμού Python

Η βασική γλώσσα υλοποίησης του συστήματος είναι η **Python**, μια υψηλού επιπέδου, δυναμικά τυποποιημένη γλώσσα προγραμματισμού με εκτεταμένη χρήση στον τομέα της ανάπτυξης διαδικτυακών εφαρμογών. Η Python προσφέρει καθαρή σύνταξη, υψηλή αναγνωσιμότητα και πλούσιο οικοσύστημα βιβλιοθηκών, χαρακτηριστικά που διευκολύνουν την ανάπτυξη και συντήρηση σύνθετων εφαρμογών.

Στο πλαίσιο της παρούσας εργασίας, η Python χρησιμοποιείται για:

- την υλοποίηση της επιχειρησιακής λογικής κρατήσεων,
- τη διαχείριση ημερομηνιών και χρονικών επικαλύψεων,
- τον έλεγχο ρόλων και δικαιωμάτων χρηστών,
- και την αλληλεπίδραση με τη βάση δεδομένων μέσω αντικειμενοστραφών μοντέλων.

Η επιλογή της Python κρίνεται ιδιαίτερα κατάλληλη για ερευνητικά έργα, καθώς επιτρέπει την εστίαση στη λογική του συστήματος μειώνοντας την συντακτική πολυπλοκότητα.

3.3 Πλαίσιο Ανάπτυξης(framework) Django

3.3.1 Γενική Περιγραφή του Django

Το Django επιλέχθηκε ως πλήρες web framework που παρέχει ενσωματωμένες λύσεις για routing, authentication, ORM, φόρμες, μηνύματα χρήστη και εργαλεία διαχείρισης. Η χρήση ενός τέτοιου framework είναι κατάλληλη για την γρήγορη παραγωγή ενός λειτουργικού πρωτοτύπου, χωρίς να υποβαθμίζεται η ποιότητα της υλοποίησης (Django Software Foundation, 2024).

Η φιλοσοφία του Django (MVT) εμφανίζεται καθαρά στην κατανομή ρόλων, τα models εκφράζουν το σχήμα και μέρος της συμπεριφοράς, τα views ορίζουν τη ροή HTTP και την πλοήγηση, ενώ η ουσιώδης επιχειρησιακή λογική συγκεντρώνεται σε services ώστε να αποφεύγεται η διάχυση κανόνων σε πολλαπλά views. Αυτή η επιλογή είναι τεχνικά σημαντική, διότι μειώνει τον κίνδυνο να υπάρχουν “διαφορετικές αλήθειες” για τον ίδιο κανόνα (π.χ. διαφορετικός έλεγχος επικάλυψης στη δημιουργία και διαφορετικός στην έγκριση) (Fowler, 2002).

Django MVT:

- τα ***Models*** αναπαριστούν τη δομή και τη λογική των δεδομένων,
- τα ***Views*** περιέχουν την επιχειρησιακή λογική και χειρίζονται τα αιτήματα,
- και τα ***Templates*** είναι υπεύθυνα για την παρουσίαση των δεδομένων στον χρήστη.

Πρακτικά, η αξιοποίηση ενσωματωμένων δυνατοτήτων του Django επιτρέπει στον κώδικα να εστιάσει στη σωστή αποτύπωση της διαθεσιμότητας των καταλλυμάτων χωρίς να χρειάζεται να φτιαχτεί από την αρχή η λογική των authentication, sessions και η βασική δρομολόγηση και ανταυτού να επικεντρωθεί στην υλοποίηση της κράτησης και των γεγονότων ημερολογίου.

3.3.2 Χρήση Django Apps και Modular Σχεδίαση

Η εφαρμογή οργανώθηκε σε διακριτά apps που αντιστοιχούν σε υποσυστήματα του προβλήματος: **core**, **accommodations**, **reservations** και **dashboard**. Η modular σχεδίαση προκύπτει τόσο από τη δρομολόγηση όσο και από το επίπεδο κώδικα, καθώς κάθε app διαθέτει δικά του views και URL namespace. Η ύπαρξη διακριτού dashboard για ιδιοκτήτη αποτελεί χαρακτηριστικό παράδειγμα "καθαρού διαχωρισμού ρόλων" όπου οι διαχειριστικές ενέργειες δεν αναμειγνύονται με τη δημόσια διεπαφή, αλλά ζουν σε αυτόνομο χώρο, με δικά του templates, δικό του routing και ξεχωριστές προϋποθέσεις πρόσβασης.

Στο επίπεδο υλοποίησης, ο διαχωρισμός αυτός κάνει ευκολότερη και την τεκμηρίωση και τον έλεγχο. Για παράδειγμα, οι ενέργειες ιδιοκτήτη (έγκριση/απόρριψη, δημιουργία blocks, αποφάσεις σε αιτήματα ακύρωσης/τροποποίησης) προστατεύονται με decorator πρόσβασης σε views του dashboard. Παράλληλα, η δημόσια ροή χρήστη για δημιουργία αιτήματος κράτησης βρίσκεται στο reservations/views.py και οδηγεί σε service create_booking_request, χωρίς να απαιτείται ο χρήστης να "περάσει" από διαχειριστικές δομές. Η αποσύνδεση αυτή μειώνει την πιθανότητα σφαλμάτων εξουσιοδότησης και δημιουργεί πιο καθαρό χάρτη ευθυνών (Evans, 2003).

Επιπλέον, η modular οργάνωση συμβάλλει στη μελλοντική επεκτασιμότητα καθώς ένα νέο app (π.χ. πληρωμές ή ειδοποιήσεις) μπορεί να προστεθεί στο routing ως νέο prefix, χωρίς να χρειαστεί να αναδιαρθρωθεί το σύνολο. Έτσι, η επιλογή Django apps δεν είναι μόνο οργανωτική πρακτική, αλλά τεχνολογικό εργαλείο που μειώνει το κόστος εξέλιξης.

- **core**: γενικές λειτουργίες, αρχική σελίδα και εγγραφή χρηστών,
- **accommodations**: διαχείριση καταλυμάτων και σχετικών πληροφοριών,

- **reservations**: μηχανισμός κρατήσεων, διαθεσιμότητας και αιτημάτων,
- **dashboard**: διεπαφή διαχείρισης για τον ιδιοκτήτη.

Η επιλογή modular σχεδίασης επιτρέπει:

- ευκολότερη κατανόηση του κώδικα,
- ανεξάρτητη εξέλιξη υποσυστημάτων,
- και μελλοντική επέκταση της εφαρμογής χωρίς εκτεταμένες τροποποιήσεις.

3.4 Βάση Δεδομένων και ORM

3.4.1 Χρήση Σχεσιακής Βάσης Δεδομένων

Το σύστημα βασίζεται σε σχεσιακή μοντελοποίηση και αξιοποιεί το Django ORM ως επίπεδο χαρτογράφησης αντικειμένων-σχέσεων. Η επιλογή ORM είναι πόλυ χρήσιμη σε εφαρμογές όπου η επιχειρησιακή λογική πρέπει να "δένεται" με συνεκτικούς περιορισμούς δεδομένων. Στην εφαρμογή, η οντότητα Booking διασυνδέεται με το κατάλυμα μέσω ForeignKey προς τα accommodations.Accommodation, ενώ ο πελάτης συσχετίζεται με το AUTH_USER_MODEL (Django Software Foundation, 2024).

Η μοντελοποίηση αυτή επιτρέπει την έκφραση σχέσεων "ένα κατάλυμα-πολλές κρατήσεις" και "ένας χρήστης-πολλές κρατήσεις" με τρόπο ελεγχόμενο και τυπικό.

Ένα τεχνικά ουσιαστικό σημείο είναι ότι το σύστημα δεν στηρίζεται μόνο σε απλές ανακτήσεις, αλλά σε queries που υλοποιούν χρονικούς περιορισμούς. Ο έλεγχος επικαλύψεων ημερομηνιών, για παράδειγμα, εφαρμόζεται μέσω φίλτρων ORM. Η συνάρτηση is_range_available καταλήγει σε queries, ώστε να εντοπίζει συγκρούσεις με υπάρχουσες κρατήσεις. Αυτό αναδεικνύει ότι η επιλογή ORM δεν είναι απλώς "ευκολία σύνταξης", αλλά μέσο για να εκφραστούν κανόνες με σαφή αντιστοίχιση σε σημασιολογία SQL, χωρίς την ανάγκη να γραφτεί χειροκίνητα SQL.

Παράλληλα, η βάση αποτυπώνει όχι μόνο την "κατάσταση", αλλά και το ιστορικό, μέσω BookingLog. Η ύπαρξη logs ως ανεξάρτητης οντότητας είναι τεχνολογικά σημαντική και

προϋποθέτει ότι το σύστημα σχεδιάστηκε για ελεγκσιμότητα (auditability) και όχι μόνο για λειτουργία.

Οι βασικές οντότητες της βάσης περιλαμβάνουν:

- καταλύματα (accommodations),
- κρατήσεις (reservations),
- περιόδους μη διαθεσιμότητας,
- αιτήματα ακύρωσης και τροποποίησης,
- και καταγραφές ενεργειών (logs).

Η επιλογή ORM διευκολύνει την επιβολή κανόνων ακεραιότητας, όπως οι συσχετίσεις μεταξύ οντοτήτων και οι περιορισμοί στις καταστάσεις κρατήσεων.

3.5 Διαχείριση Χρηστών και Ρόλων

Η εφαρμογή αξιοποιεί το ενσωματωμένο σύστημα authentication/authorization του Django και το επεκτείνει με ρόλους που βασίζονται σε groups. Στο dashboard, οι view συναρτήσεις προστατεύονται με decorator που απαιτεί ιδιοκτήτη, εξασφαλίζοντας ότι λειτουργίες υψηλού κινδύνου (όπως έγκριση/απόρριψη ή δημιουργία blocks) δεν είναι προσβάσιμες από απλούς χρήστες (Django Software Foundation, 2024).

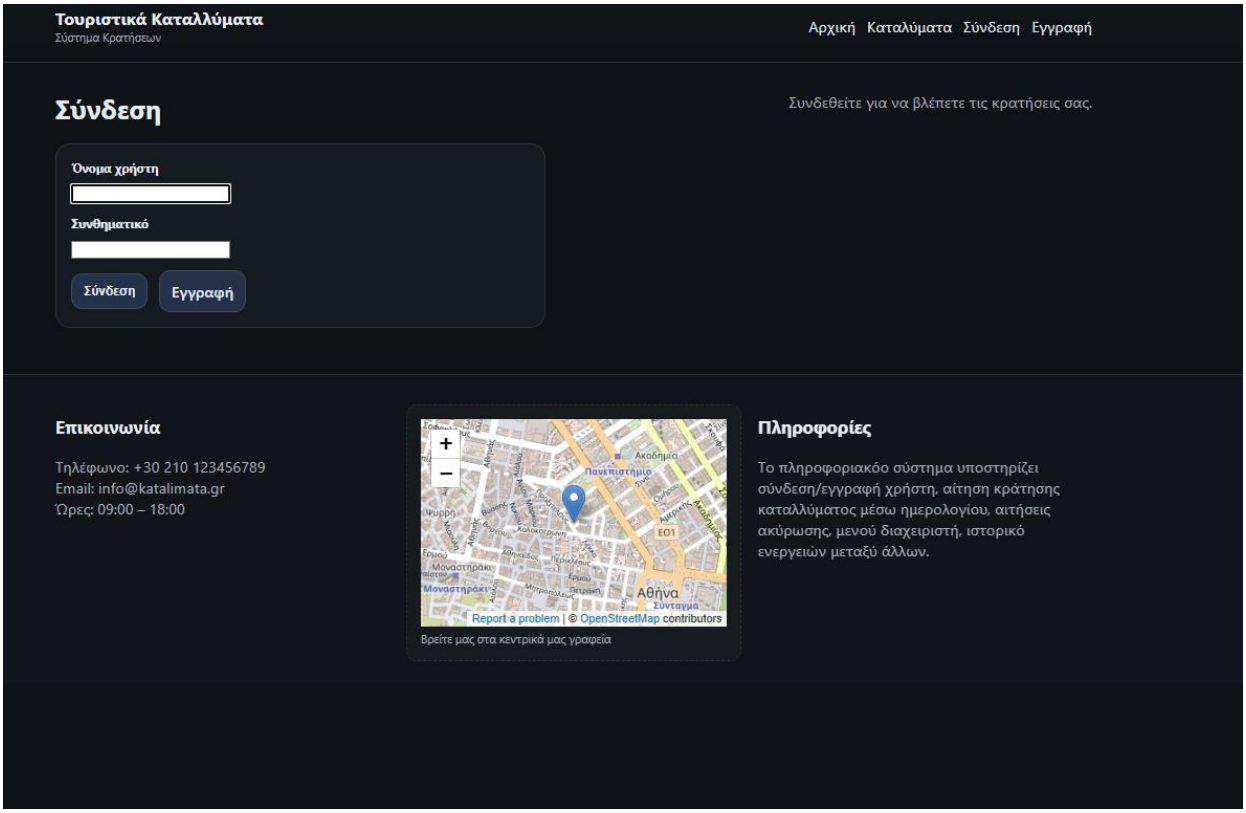
Η επιλογή groups αντί για ένα πεδίο ρόλου στο user model έχει πρακτικές συνέπειες. Αρχικά επιτρέπει εύκολη αλλαγή πολιτικής (π.χ. αλλαγή ονόματος ομάδας μέσω OWNER_GROUP_NAME setting). Αυτή η παραμετροποίηση είναι σημαντική, γιατί επιτρέπει στο σύστημα να προσαρμοστεί σε διαφορετικά περιβάλλοντα χωρίς τροποποίηση επιχειρησιακής λογικής. Παράλληλα, η ασφάλεια δεν στηρίζεται μόνο στον έλεγχο “σελίδας”, αλλά και στον έλεγχο δεδομένων. Η πρόσβαση σε κρατήσεις πελάτη, για παράδειγμα, υλοποιείται με queries που φιλτράρουν με βάση τον customer, μειώνοντας τον κίνδυνο ανάγνωσης ξένων εγγραφών. Η ασφάλεια αντιμετωπίζεται ως ιδιότητα του backend, όχι ως UI περιορισμός.

Ο έλεγχος ρόλων υλοποιείται μέσω:

- ομάδων χρηστών (groups),
- decorators που περιορίζουν την πρόσβαση σε συγκεκριμένες προβολές,
- και ελέγχων σε επίπεδο επιχειρησιακής λογικής.

Η επιλογή αυτή εξασφαλίζει σαφή διαχωρισμό αρμοδιοτήτων και αυξημένη ασφάλεια, χωρίς την ανάγκη εξωτερικών βιβλιοθηκών.

Η οθόνη σύνδεσης, μέσω της οποίας ο χρήστης αποκτά ταυτότητα στο σύστημα και, ανάλογα με την ομάδα στην οποία ανήκει, πρόσβαση στις αντίστοιχες λειτουργίες, παρουσιάζεται στην Εικόνα 4.



Εικόνα 4 Οθόνη σύνδεσης χρήστη στο σύστημα

3.6 Τεχνολογίες Διεπαφής Χρήστη

3.6.1 HTML Templates και Django Templating Engine

Η διεπαφή υλοποιείται με server-rendered HTML templates, αξιοποιώντας τον Django templating engine για δυναμική παραγωγή περιεχομένου. Το σύστημα περιέχει εκτενή templates όπως “Dashboard Booking” και σελίδες διαχείρισης blocks, που χρησιμοποιούν τυπικούς μηχανισμούς του Django. Η χρήση των templates συνδέεται άμεσα με τις τεχνολογικές αποφάσεις του backend, το ίδιο επιχειρησιακό αποτέλεσμα αποτυπώνεται ως οδηγία χρήσης στη διεπαφή, ενώ η πραγματική επιβολή γίνεται στις υπηρεσίες και τα queries. Ιδιαίτερα σημαντική είναι η χρήση προστασίας CSRF, που αποτελεί ενσωματωμένο χαρακτηριστικό του Django και εμφανίζεται ρητά στα templates σε φόρμες POST. Η τεχνολογική αυτή επιλογήυπάρχει επειδή το σύστημα περιλαμβάνει ενέργειες αλλαγής κατάστασης (έγκριση, απόρριψη, δημιουργία block), η CSRF προστασία είναι προϋπόθεση ασφαλούς λειτουργίας (Django Software Foundation, 2024).

3.6.2 CSS και Οπτική Ταυτότητα

Η μορφοποίηση γίνεται μέσω στατικού CSS αρχείου (static/css/app.css), το οποίο περιλαμβάνει ορισμό μεταβλητών CSS καθώς και κλάσεις για πλοήγηση, κουμπιά και γενικότερα για την τυπογραφία. Η επιλογή ενός συνεκτικού CSS layer, χωρίς υπερβολικά εξωτερικά frameworks, εξυπηρετεί δύο τεχνολογικούς στόχους, αρχικά, διατηρεί χαμηλό το βάρος εξαρτήσεων, και δεύτερον, επιτρέπει να αποτυπωθούν οι καταστάσεις του συστήματος με σαφείς οπτικές συμβάσεις. Παρότι η οπτική σχεδίαση δεν είναι ο κύριος στόχος της εργασίας, η ύπαρξη του αρχείου CSS βελτιώνει τη χρηστικότητα και τη δομική κατανόηση της διεπαφής. Ένα σύστημα κρατήσεων πρέπει να επικοινωνεί καθαρά την κατάσταση, και η τεχνολογική επιλογή του CSS υποστηρίζει ακριβώς αυτή την ανάγκη.

3.7 Εργαλεία Ανάπτυξης και Περιβάλλον Εργασίας

Η ανάπτυξη πραγματοποιήθηκε σε τυπικό Django περιβάλλον, αξιοποιώντας τον development server και τη δυνατότητα τοπικής εξυπηρέτησης στατικών αρχείων σε συνθήκες debugging. Το

Django παρέχει το admin ως εργαλείο διαχείρισης δεδομένων. Αν και το admin δεν αποτυπώνεται ως UI στόχος της εργασίας, η ενεργοποίησή του στο routing (path("admin/", admin.site.urls)) αποτελεί τεχνολογική επιλογή που διευκολύνει την επαλήθευση του σχήματος και τη διαγνωστική παρατήρηση δεδομένων. Καθώς το σύστημα διαθέτει σύνθετες καταστάσεις (κρατήσεις, αιτήματα κτλπ), η δυνατότητα άμεσης επιθεώρησης της βάσης μέσω admin είναι σημαντική για debugging και για έλεγχο ορθότητας των δεδομένων.

Επίσης έχει δημιουργηθεί script δημιουργίας δεδομένων για καταλύματ, εικονες, χρήστες και κρατήσεις. Αυτό αποτελεί εργαλείο ανάπτυξης και επιτρέπει να δημιουργηθούν δεδομένα χωρίς χειροκίνητη εισαγωγή.

3.8 Συνολική Αξιολόγηση Τεχνολογικών Επιλογών

Συνολικά, ο συνδυασμός Python και Django αποδεικνύεται κατάλληλος για το συγκεκριμένο πρόβλημα, διότι επιτρέπει σαφή μοντελοποίηση εννοιών όπως “κράτηση”, “block”, “hold” και “κατάσταση”, και μεταφέρει την κρίσιμη λογική στο backend, όπου μπορεί να ελεγχθεί και να προστατευτεί. Η ύπαρξη κεντρικών υπηρεσιών στο reservations/services.py δείχνει τεχνολογική ωριμότητα καθώς οι κανόνες διαθεσιμότητας και οι μεταβάσεις κατάστασης δεν είναι διάσπαρτες σε views, αλλά συγκεντρωμένες και επαναχρησιμοποιήσιμες.

Παράλληλα, η αξιοποίηση ORM και συναλλαγών σε ενέργειες έγκρισης/απόρριψης, ενισχύει την αξιοπιστία και την ακεραιότητα, που αποτελούν βασικές μη λειτουργικές απαιτήσεις. Η τεχνολογική επιλογή να καταγράφονται ενέργειες στο BookingLog προσθέτει ένα επίπεδο auditability που είναι ιδιαίτερα χρήσιμο στο συγκεκριμένο σύστημα όπου ένας διαχειριστής είναι υπεύθυνος για όλες τις κινήσεις που γίνονται στην εφαρμογή. Τέλος, τα templates και το CSS παρέχουν ένα πλήρες web περιβάλλον, με σαφή ροή χρήσης και προστατευμένες ενέργειες μέσω CSRF.

Ως αποτέλεσμα, οι τεχνολογικές επιλογές δεν λειτουργούν αποσπασματικά, αλλά συγκροτούν ένα συνεκτικό οικοσύστημα εργαλείων που εξυπηρετεί τόσο την υλοποίηση των απαιτήσεων όσο και την τεκμηρίωση της εργασίας.

ΚΕΦΑΛΑΙΟ 4ο

4. ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΣΥΣΤΗΜΑΤΟΣ

4.1 Γενική Αρχιτεκτονική Προσέγγιση

Η αρχιτεκτονική ενός πληροφοριακού συστήματος καθορίζει σε μεγάλο βαθμό την ποιότητα, τη συντηρησιμότητα και τη δυνατότητα μελλοντικής επέκτασής του. Στην παρούσα εργασία, η αρχιτεκτονική του συστήματος κρατήσεων σχεδιάστηκε με στόχο τη σαφή οργάνωση της λειτουργικότητας, τον διαχωρισμό ευθυνών και την αξιόπιστη υλοποίηση της επιχειρησιακής λογικής που διέπει τις κρατήσεις τουριστικών καταλυμάτων. Το σύστημα υλοποιήθηκε ως μονολιθική διαδικτυακή εφαρμογή (monolithic web application) βασισμένη στο πλαίσιο Django, ακολουθώντας την αρχιτεκτονική Model-View-Template (MVT). Η επιλογή της μονολιθικής προσέγγισης κρίθηκε κατάλληλη για το εύρος και τη φύση του έργου, καθώς επιτρέπει την κεντρική διαχείριση της λογικής και των δεδομένων, χωρίς την πολυπλοκότητα που συνεπάγονται κατανεμημένες ή μικροϋπηρεσιακές αρχιτεκτονικές (Fowler, 2002).

Παρά τον μονολιθικό χαρακτήρα της εφαρμογής, η εσωτερική της δομή είναι αυστηρά modular, με σαφή διαχωρισμό υποσυστημάτων μέσω ανεξάρτητων Django apps. Με τον τρόπο αυτό επιτυγχάνεται ισορροπία μεταξύ απλότητας και επεκτασιμότητας.

4.2 Δομή Εφαρμογής και Υποσυστήματα

Η συνολική αρχιτεκτονική της εφαρμογής διαρθρώνεται σε διακριτά υποσυστήματα, καθένα από τα οποία εξυπηρετεί συγκεκριμένες λειτουργικές απαιτήσεις. Ο διαχωρισμός αυτός δεν είναι μόνο εννοιολογικός, αλλά αποτυπώνεται ξεκάθαρα στη δομή του κώδικα.

4.2.1 Υποσύστημα Δημόσιας Διεπαφής

Το υποσύστημα δημόσιας διεπαφής αφορά όλες τις λειτουργίες που είναι προσβάσιμες από ανώνυμους ή μη αυθεντικοποιημένους χρήστες. Περιλαμβάνει την αρχική σελίδα της εφαρμογής, την παρουσίαση των καταλυμάτων και την προβολή της διαθεσιμότητας.

Αρχιτεκτονικά, το υποσύστημα αυτό:

- λαμβάνει δεδομένα από το επίπεδο μοντέλων (καταλύματα, κρατήσεις, blocks),
- επεξεργάζεται τη λογική εμφάνισης της διαθεσιμότητας στο επίπεδο των views,
- και αποδίδει τα αποτελέσματα μέσω HTML templates.

Ιδιαίτερη σημασία έχει ο μηχανισμός υπολογισμού της διαθεσιμότητας, ο οποίος εκτελείται δυναμικά κάθε φορά που φορτώνεται η αρχική σελίδα. Με τον τρόπο αυτό, η δημόσια διεπαφή δεν βασίζεται σε αποθηκευμένα ή στατικά δεδομένα, αλλά αντικατοπτρίζει πάντοτε την πραγματική κατάσταση του συστήματος.

4.2.2 Υποσύστημα Κρατήσεων

Το υποσύστημα κρατήσεων αποτελεί τον **πυρήνα της αρχιτεκτονικής** και συγκεντρώνει τη βασική επιχειρησιακή λογική της εφαρμογής. Περιλαμβάνει τη δημιουργία αιτημάτων κράτησης, τον έλεγχο διαθεσιμότητας, τη διαχείριση καταστάσεων και τη χρονική δέσμευση ημερομηνιών.

Αρχιτεκτονικά, το υποσύστημα αυτό διαχωρίζεται σε τρία επίπεδα:

1. **Επίπεδο Μοντέλων (Models):** Περιλαμβάνει οντότητες όπως η κράτηση, οι περίοδοι μη διαθεσιμότητας και τα αιτήματα τροποποίησης ή ακύρωσης. Κάθε οντότητα ενσωματώνει συγκεκριμένους κανόνες και καταστάσεις, οι οποίες αντικατοπτρίζουν τον κύκλο ζωής μιας κράτησης.
2. **Επίπεδο Υπηρεσιών (Services):** Η επιχειρησιακή λογική δεν υλοποιείται απευθείας στα views, αλλά μεταφέρεται σε ξεχωριστές συναρτήσεις υπηρεσιών. Αυτές είναι υπεύθυνες για:
    - τον έλεγχο επικαλύψεων ημερομηνιών,

- τη δημιουργία και λήξη προσωρινών δεσμεύσεων (holds),
- την έγκριση ή απόρριψη κρατήσεων,
- και τη διαχείριση αιτημάτων αλλαγής.

3. **Επίπεδο Προβολών (Views):** Τα views λειτουργούν ως διαμεσολαβητές μεταξύ του χρήστη και της επιχειρησιακής λογικής. Αναλαμβάνουν την παραλαβή των αιτημάτων HTTP, την κλήση των αντίστοιχων υπηρεσιών και την επιστροφή κατάλληλων αποκρίσεων ή ανακατευθύνσεων.

Η ύπαρξη σαφούς διαχωρισμού μεταξύ των παραπάνω επιπέδων αποτελεί βασικό αρχιτεκτονικό χαρακτηριστικό του συστήματος.

4.2.3 Υποσύστημα Διαχείρισης (Admin Dashboard)

Το υποσύστημα διαχείρισης απευθύνεται αποκλειστικά στον ιδιοκτήτη ή διαχειριστή των καταλυμάτων. Παρέχει συγκεντρωτική εικόνα της κατάστασης του συστήματος και εργαλεία για τη λήψη αποφάσεων.

Οι βασικές λειτουργίες του υποσυστήματος περιλαμβάνουν:

- προβολή εκκρεμών αιτημάτων κράτησης,
- έγκριση ή απόρριψη αιτημάτων,
- διαχείριση αιτημάτων ακύρωσης και τροποποίησης,
- δημιουργία και διαγραφή περιόδων μη διαθεσιμότητας,
- και ημερολογιακή απεικόνιση της συνολικής κατάστασης.

Αρχιτεκτονικά, το dashboard βασίζεται σε μηχανισμούς ελέγχου πρόσβασης που εξασφαλίζουν ότι μόνο εξουσιοδοτημένοι χρήστες μπορούν να αλληλεπιδρούν με τις λειτουργίες αυτές. Η επιχειρησιακή λογική παραμένει στο επίπεδο υπηρεσιών, ενώ το dashboard λειτουργεί ως εξειδικευμένη διεπαφή πάνω από αυτή.

4.3 Ροές Δεδομένων και Επιχειρησιακή Λογική

Η ροή δεδομένων στο σύστημα ακολουθεί προκαθορισμένα μονοπάτια, τα οποία εξασφαλίζουν συνέπεια και αξιοπιστία. Ενδεικτικά, η ροή ενός αιτήματος κράτησης περιλαμβάνει τα εξής στάδια:

1. Υποβολή δεδομένων από τον χρήστη μέσω φόρμας.
2. Παραλαβή και αρχικός έλεγχος εγκυρότητας στο επίπεδο view.
3. Κλήση της υπηρεσίας δημιουργίας κράτησης.
4. Έλεγχος διαθεσιμότητας και επικαλύψεων.
5. Δημιουργία εγγραφής κράτησης σε κατάσταση εκκρεμής.
6. Ενημέρωση του συστήματος μέσω καταγραφής ενέργειας.

Αντίστοιχες ροές εφαρμόζονται για την έγκριση, απόρριψη, ακύρωση και τροποποίηση κρατήσεων. Η τυποποίηση των ροών αυτών μειώνει την πιθανότητα σφαλμάτων και διευκολύνει τον έλεγχο της ορθότητας του συστήματος.

4.4 Διαχείριση Συνέπειας και Συναλλαγών

Ιδιαίτερη έμφαση δόθηκε στη διαχείριση της συνέπειας των δεδομένων. Για τον λόγο αυτό, κρίσιμες λειτουργίες υλοποιούνται εντός συναλλαγών (transactions), ώστε να διασφαλίζεται ότι είτε θα ολοκληρώνονται πλήρως είτε δεν θα εφαρμόζονται καθόλου (Django Software Foundation, 2024, Bernstein, Hadzilacos & Goodman, 1987).

Η χρήση συναλλαγών:

- αποτρέπει τη δημιουργία αντικρουόμενων κρατήσεων,
- διασφαλίζει τη σωστή μετάβαση καταστάσεων,
- και προστατεύει το σύστημα από μερικές ή ασυνεπείς ενημερώσεις δεδομένων.

Επιπλέον, η ύπαρξη μηχανισμού καταγραφής ενεργειών (logging) αποτελεί στοιχείο σημαντικό τόσο για λόγους ελέγχου όσο και για μελλοντική ανάλυση.

ΚΕΦΑΛΑΙΟ 5ο

5. ΣΧΕΔΙΑΣΗ ΒΑΣΗΣ ΔΕΔΟΜΕΝΩΝ

5.1 Εισαγωγή στη Σχεδίαση της Βάσης Δεδομένων

Η σχεδίαση της βάσης δεδομένων αποτελεί θεμελιώδες στάδιο στην ανάπτυξη πληροφοριακών συστημάτων, καθώς καθορίζει τον τρόπο με τον οποίο αποθηκεύονται, συσχετίζονται και διαχειρίζονται τα δεδομένα.

Στο παρόν σύστημα, η βάση δεδομένων σχεδιάστηκε με στόχο:

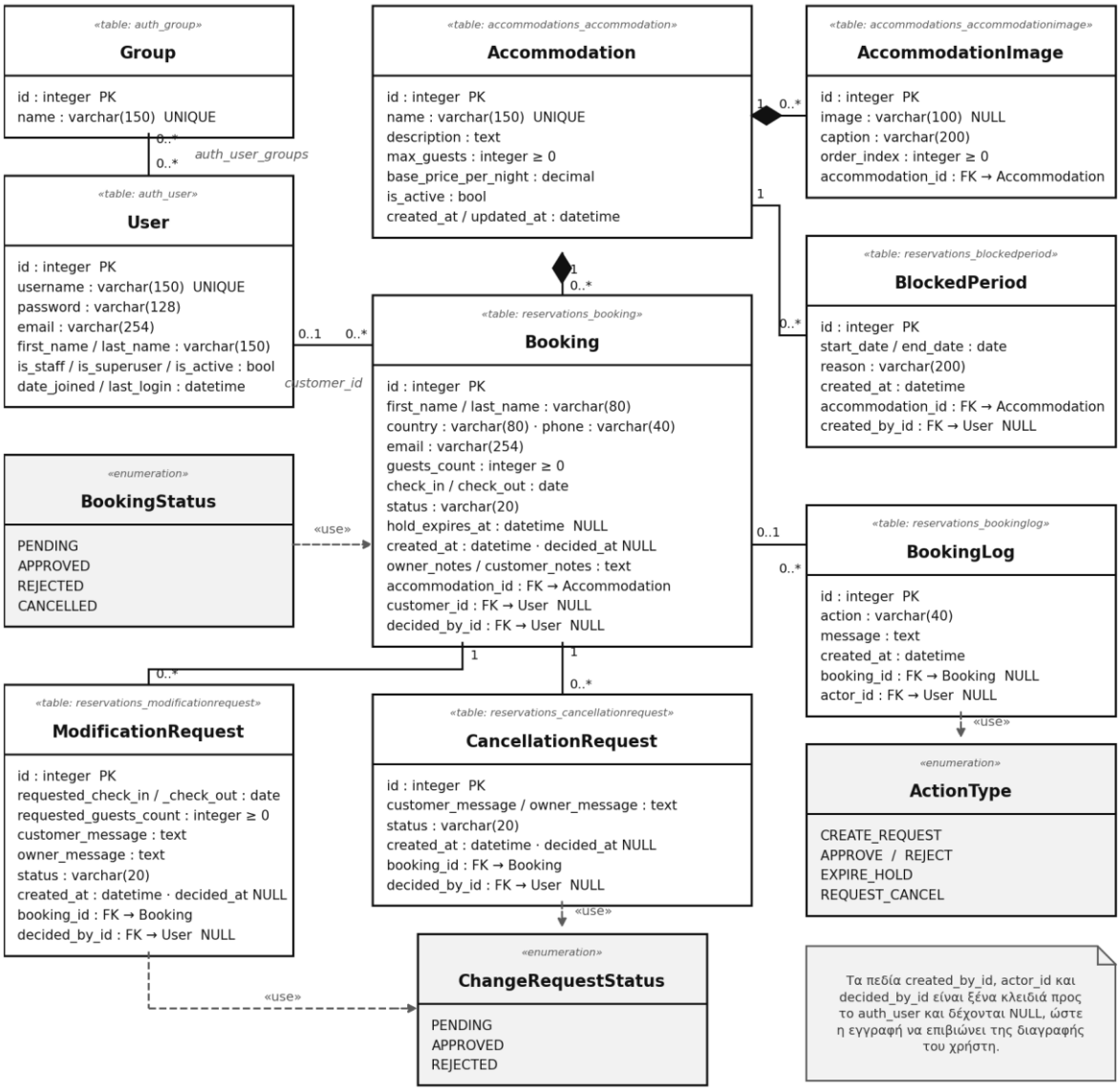
- τη σαφή μοντελοποίηση των βασικών εννοιών του συστήματος κρατήσεων,
- τη διασφάλιση της ακεραιότητας και της συνέπειας των δεδομένων,
- την αποφυγή επικαλύψεων ημερομηνιών,
- και την υποστήριξη των ροών εργασίας που περιγράφονται στα προηγούμενα κεφάλαια.

Η υλοποίηση της βάσης δεδομένων πραγματοποιήθηκε μέσω του Django ORM, το οποίο επιτρέπει τη μετατροπή των εννοιολογικών οντοτήτων σε σχεσιακούς πίνακες με σαφώς ορισμένα πεδία και συσχετίσεις (Django Software Foundation, 2024).

5.2 Ανάλυση Οντοτήτων και Συσχετίσεων

Η σχεδίαση της βάσης δεδομένων βασίζεται σε ένα σύνολο βασικών οντοτήτων, καθεμία από τις οποίες αντιστοιχεί σε συγκεκριμένη επιχειρησιακή έννοια. Οι οντότητες αυτές συνδέονται μεταξύ τους μέσω συσχετίσεων τύπου ένα-προς-πολλά ή πολλά-προς-ένα, σύμφωνα με τις ανάγκες του συστήματος.

Το σύνολο των οντοτήτων και των μεταξύ τους συσχετίσεων αποτυπώνεται στο διάγραμμα κλάσεων της Εικόνας 5, το οποίο αντιστοιχεί άμεσα στα μοντέλα (models) της εφαρμογής. Στο διάγραμμα φαίνονται οι σχέσεις σύνθεσης του καταλύματος με τις κρατήσεις και τις περιόδους μη διαθεσιμότητας, η συσχέτιση του χρήστη με τις κρατήσεις του, οι οντότητες αιτημάτων ακύρωσης και τροποποίησης, η οντότητα καταγραφής ενεργειών, καθώς και τα enumerations που ορίζουν τις επιτρεπτές τιμές κατάστασης.



Εικόνα 5 Διάγραμμα κλάσεων των οντοτήτων της βάσης δεδομένων

5.2.1 Οντότητα Καταλύματος

Η οντότητα *Κατάλυμα* (accommodation) αποτελεί τον πυρήνα της πληροφορίας του συστήματος. Κάθε κατάλυμα αντιπροσωπεύει ένα ανεξάρτητο δωμάτιο ή διαμέρισμα που μπορεί να κρατηθεί από τους χρήστες. Τα βασικά γνωρίσματα της οντότητας περιλαμβάνουν:

- μοναδικό αναγνωριστικό,
- ονομασία,
- περιγραφή,
- μέγιστο αριθμό φιλοξενούμενων,
- βασική τιμή ανά διανυκτέρευση,
- και κατάσταση ενεργοποίησης.

Η ύπαρξη κατάστασης ενεργοποίησης επιτρέπει την απόσυρση ενός καταλύματος από τη δημόσια προβολή χωρίς διαγραφή των ιστορικών δεδομένων κρατήσεων.

Κάθε κατάλυμα συσχετίζεται με πολλαπλές κρατήσεις και πολλαπλές περιόδους μη διαθεσιμότητας, γεγονός που αντικατοπτρίζεται σε συσχετίσεις *ένα-προς-πολλά*.

5.2.2 Οντότητα Κράτησης

Η οντότητα *Κράτηση* (reservation) αποτελεί τη σημαντικότερη οντότητα του συστήματος, καθώς αποτυπώνει τη σχέση μεταξύ ενός καταλύματος και ενός πελάτη για συγκεκριμένο χρονικό διάστημα. Η σχεδίασή της έγινε με ιδιαίτερη προσοχή, ώστε να υποστηρίζεται πλήρως ο κύκλος ζωής μιας κράτησης.

Τα βασικά πεδία της οντότητας περιλαμβάνουν:

- αναφορά στο κατάλυμα,
- αναφορά στον χρήστη (εφόσον υπάρχει),
- προσωπικά στοιχεία πελάτη (αποθηκευμένα ως στιγμιότυπο),

- ημερομηνία άφιξης (check-in),
- ημερομηνία αναχώρησης (check-out),
- αριθμό φιλοξενούμενων,
- κατάσταση κράτησης,
- χρονικό όριο προσωρινής δέσμευσης (hold),
- χρονικές σφραγίδες δημιουργίας και απόφασης.

Η αποθήκευση των προσωπικών στοιχείων ως ανεξάρτητα πεδία και όχι μέσω απλής αναφοράς στον χρήστη επιτρέπει τη διατήρηση ιστορικής ακρίβειας, ακόμη και αν τα στοιχεία του λογαριασμού αλλάξουν μελλοντικά.

5.2.3 Καταστάσεις και Κύκλος Ζωής Κράτησης

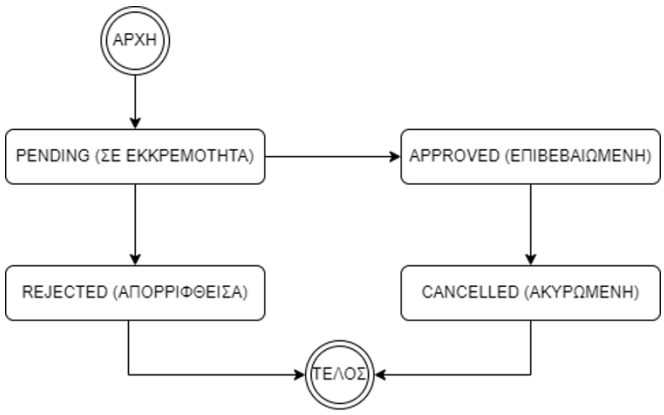
Η κατάσταση μιας κράτησης αποτελεί κρίσιμο στοιχείο της σχεδίασης. Στο σύστημα ορίζονται διακριτές καταστάσεις, όπως:

- εκκρεμής,
- εγκεκριμένη,
- απορριφθείσα,
- ακυρωμένη.

Η μετάβαση μεταξύ των καταστάσεων αυτών ακολουθεί αυστηρά καθορισμένους κανόνες. Για παράδειγμα, μόνο μια εκκρεμής κράτηση μπορεί να εγκριθεί ή να απορριφθεί, ενώ μόνο μια εγκεκριμένη κράτηση μπορεί να τροποποιηθεί ή να αποτελέσει αντικείμενο αιτήματος ακύρωσης.

Η σχεδίαση αυτή διασφαλίζει ότι η βάση δεδομένων δεν μπορεί να περιέχει λογικά ασυνεπείς εγγραφές.

Η ακολουθία των επιτρεπτών μεταβάσεων παρουσιάζεται στο διάγραμμα καταστάσεων της Εικόνας 6.



Εικόνα 6 Διάγραμμα καταστάσεων του κύκλου ζωής μιας κράτησης

5.2.4 Οντότητα Περιόδου Μη Διαθεσιμότητας

Η οντότητα *Περίοδος Μη Διαθεσιμότητας* (blocked period) χρησιμοποιείται για την αποτύπωση χρονικών διαστημάτων κατά τα οποία ένα κατάλυμα δεν μπορεί να κρατηθεί, ανεξάρτητα από κρατήσεις πελατών. Τέτοιες περίοδοι μπορεί να οφείλονται σε συντήρηση, ιδιοχρησία ή άλλους λόγους.

Κάθε εγγραφή περιλαμβάνει:

- αναφορά στο κατάλυμα,
- ημερομηνία έναρξης,
- ημερομηνία λήξης,
- προαιρετική αιτιολογία,
- και στοιχεία δημιουργίας.

Η ύπαρξη αυτής της οντότητας επιτρέπει την ευέλικτη διαχείριση διαθεσιμότητας χωρίς την ανάγκη δημιουργίας ψευδών κρατήσεων.

5.2.5 Οντότητες Αιτημάτων Τροποποίησης και Ακύρωσης

Για την υποστήριξη πιο σύνθετων σεναρίων χρήσης, το σύστημα περιλαμβάνει ξεχωριστές οντότητες για:

- αιτήματα ακύρωσης,
- και αιτήματα τροποποίησης κράτησης.

Οι οντότητες αυτές συνδέονται άμεσα με μια εγκεκριμένη κράτηση και διαθέτουν δικό τους κύκλο ζωής, με καταστάσεις όπως εκκρεμής, εγκεκριμένη ή απορριφθείσα. Με τον τρόπο αυτό, η βάση δεδομένων μπορεί να αποτυπώνει με σαφήνεια τόσο το ιστορικό των αιτημάτων όσο και τις αποφάσεις του ιδιοκτήτη.

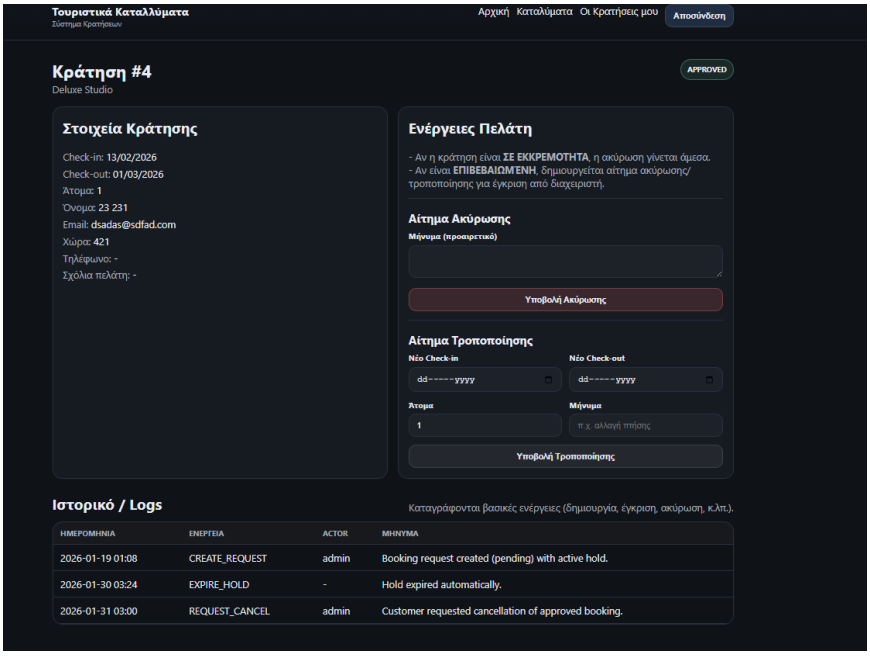
5.2.6 Οντότητα Καταγραφής Ενεργειών

Η οντότητα καταγραφής ενεργειών (log) χρησιμοποιείται για την αποθήκευση σημαντικών συμβάντων του συστήματος, όπως δημιουργία κράτησης, έγκριση, απόρριψη, ακύρωση ή δημιουργία περιόδου μη διαθεσιμότητας.

Η ύπαρξη αυτής της οντότητας:

- ενισχύει τη διαφάνεια,
- διευκολύνει την αποσφαλμάτωση,
- και προσφέρει πλήρη ιστορικότητα των ενεργειών.

Η Εικόνα 7 παρουσιάζει το ιστορικό μιας κράτησης όπως αποδίδεται στη διεπαφή, με τις καταγεγραμμένες ενέργειες CREATE_REQUEST, EXPIRE_HOLD και REQUEST_CANCEL, τον χρήστη που τις προκάλεσε και τη χρονική τους σήμανση.



Εικόνα 7 Μενού - Λεπτομέρειες κράτησης, δυνατότητα υποβολής αιτήματος ακύρωσης, προβολής στοιχείων κράτησης και προβολή ιστορικού

5.3 Κανόνες Ακεραιότητας και Επιχειρησιακοί Περιορισμοί

Η βάση δεδομένων δεν λειτουργεί απλώς ως αποθετήριο δεδομένων, αλλά ενσωματώνει βασικούς κανόνες ακεραιότητας. Ενδεικτικά, δεν επιτρέπει την ημερομηνία αναχώρησης να είναι προγενέστερη της ημερομηνίας άφιξης, δεν επιτρέπεται η υπέρβαση του μέγιστου αριθμού φιλοξενούμενων για το εκάστοτε δωμάτιο και δεν επιτρέπονται επικαλύψεις ημερομηνιών μεταξύ εγκεκριμένων κρατήσεων και περιόδων μη διαθεσιμότητας.

Οι κανόνες αυτοί υλοποιούνται συνδυαστικά, σε επίπεδο βάσης δεδομένων μέσω των περιορισμών αλλά και σε επίπεδο εφαρμογής μέσω ελέγχων στην επιχειρησιακή λογική.

5.4 Διαχείριση Επικάλυψης Ημερομηνιών

Ιδιαίτερη πρόκληση στη σχεδίαση της βάσης δεδομένων αποτελεί η αποφυγή επικαλύψεων ημερομηνιών. Το σύστημα υιοθετεί σαφή σημασιολογία χρονικών διαστημάτων, όπου η ημερομηνία άφιξης περιλαμβάνεται στο διάστημα, ενώ η ημερομηνία αναχώρησης εξαιρείται. Η επιλογή αυτή απλοποιεί τους ελέγχους επικαλύψεων και επιτρέπει την ακριβή διαδοχή κρατήσεων χωρίς κενά ή επικαλύψεις (Allen, 1983).

ΚΕΦΑΛΑΙΟ 6ο

6. ΥΛΟΠΟΙΗΣΗ ΣΥΣΤΗΜΑΤΟΣ

6.1 Εισαγωγή στην Υλοποίηση

Η υλοποίηση του συστήματος κρατήσεων αποτελεί το στάδιο κατά το οποίο οι απαιτήσεις και οι αρχιτεκτονικές επιλογές μετασχηματίζονται σε εκτελέσιμο λογισμικό, με ελέγξιμη συμπεριφορά και σαφώς ορισμένες ροές εργασίας. Σε ένα περιβάλλον τουριστικών κρατήσεων, ο πυρήνας της πολυπλοκότητας δεν εντοπίζεται στην απλή αποθήκευση δεδομένων, αλλά στη διασφάλιση επιχειρησιακών κανόνων όπως η αποφυγή χρονικών επικαλύψεων, η ορθή μετάβαση καταστάσεων (π.χ. από εκκρεμή σε εγκεκριμένη), και η πρόληψη ασυνέπειας όταν πολλαπλά αιτήματα αλληλεπιδρούν με το ίδιο κατάλυμα σε κοντινό χρονικό διάστημα. Στον κώδικα του έργου, η πολυπλοκότητα αυτή αντιμετωπίζεται με έναν συνδυασμό δομικής οργάνωσης (modular apps), αυστηρής μοντελοποίησης (models) και κεντρικοποίησης επιχειρησιακής λογικής (services) (Evans, 2003).

Η υλοποίηση βασίζεται στο Django και στη λογική MVT, ωστόσο το έργο αποφεύγει την υπερσυγκέντρωση λογικής στα views. Αντίθετα, οι κρίσιμες λειτουργίες μεταφέρονται στο επίπεδο υπηρεσιών (reservations/services.py), όπου εφαρμόζονται αλγόριθμοι διαθεσιμότητας, μηχανισμοί holds και συναλλαγές. Η επιλογή αυτή είναι ουσιώδης για τη συντηρησιμότητα, διότι επιτρέπει στα views να παραμένουν κυρίως ως διαχειριστές HTTP αιτημάτων/αποκρίσεων, ενώ οι κανόνες που καθορίζουν τη «σωστή» λειτουργία του συστήματος βρίσκονται σε επαναχρησιμοποιήσιμες συναρτήσεις που μπορούν να κληθούν από διαφορετικά σημεία (δημόσια διεπαφή, dashboard, API).

6.2 Δρομολόγηση και Διαχείριση URLs

Η δρομολόγηση των αιτημάτων HTTP υλοποιείται μέσω του κεντρικού αρχείου booking_site/booking_site/urls.py, το οποίο λειτουργεί ως σημείο ενοποίησης των επιμέρους υποσυστημάτων. Το έργο δεν «φορτώνει» ένα μονολιθικό σύνολο διαδρομών, αλλά χρησιμοποιεί include() για να αναθέσει τη διαχείριση διαδρομών στα αντίστοιχα apps. Έτσι, η βασική δομή επιτυγχάνει θεματική απομόνωση, διαφορετικό υποσύστημα για κρατήσεις (reservations), διαφορετικό για διαχείριση (dashboard), και διαφορετικό για τον πυρήνα/landing λειτουργίες (core).

Αρχικά μειώνεται η σύζευξη μεταξύ λειτουργιών, διότι αλλαγές στις διαδρομές κρατήσεων περιορίζονται στο reservations/urls.py χωρίς να επηρεάζουν το dashboard ή τις σελίδες πλοήγησης και ενισχύεται η διαγνωσιμότητα καθώς ο προγραμματιστής μπορεί να εντοπίσει γρήγορα πού ορίζεται ένα endpoint, άρα και πού πρέπει να εξεταστεί η αντίστοιχη υλοποίηση.

Το στοιχείο αυτό αποκτά ιδιαίτερη σημασία σε συστήματα κρατήσεων, όπου η δημόσια διεπαφή, οι προσωπικές σελίδες χρήστη (π.χ. “my bookings”) και οι λειτουργίες ιδιοκτήτη πρέπει να παραμένουν σαφώς διαχωρισμένες και να προστατεύονται από ακούσια «διαρροή» δικαιωμάτων.

Σε επίπεδο υλοποίησης, η διάκριση αυτή φαίνεται και στον τρόπο με τον οποίο χαρτογραφούνται οι λειτουργίες. Για παράδειγμα, το reservations/views.py περιλαμβάνει endpoints που σχετίζονται με δημιουργία αιτήματος κράτησης, προβολή κρατήσεων πελάτη και API διαθεσιμότητας, ενώ το dashboard/views.py περιλαμβάνει endpoints για έγκριση/απόρριψη, δημιουργία blocks και διαχειριστική προβολή. Με αυτόν τον τρόπο, η δρομολόγηση δεν είναι απλώς «τεχνική ρύθμιση», αλλά μηχανισμός ενίσχυσης της αρχιτεκτονικής καθαρότητας.

6.3 Υλοποίηση Δημόσιας Διεπαφής

Η δημόσια διεπαφή υλοποιείται κυρίως στο booking_site/core/views.py και αποτελεί το σημείο εισόδου του χρήστη στην εφαρμογή. Στον συγκεκριμένο κώδικα, παρατηρείται ότι το σύστημα δεν περιορίζεται στην απλή ανάκτηση λίστας καταλυμάτων, αλλά προχωρά σε υπολογισμό και σύνθεση πληροφορίας που σχετίζεται με διαθεσιμότητα και οπτική αναπαράσταση ημερολογίου.

Ο υπολογισμός αυτός μετατρέπει ακατέργαστες εγγραφές (κρατήσεις, blocks) σε ημερολογιακή «σηματοδότηση» ημερών.

Ο μηχανισμός υπολογισμού ημερών διαθεσιμότητας στο core/views.py αξιοποιεί βοηθητικές συναρτήσεις όπως _days_inclusive(start, end) και χειρίζεται προσεκτικά τα άκρα του μήνα. Ενδεικτικό τμήμα λογικής είναι ο μετασχηματισμός του check_out από αποκλειστικό άκρο σε «τελευταία κατειλημμένη ημέρα» μέσω της end_exclusive - timedelta(days=1). Αυτή η επιλογή αποτελεί αναγκαία προσαρμογή ώστε το ημερολόγιο (που εμφανίζει ημέρες ως διακριτά κελιά) να αντιστοιχεί σωστά στη σημασιολογία [check_in, check_out) που τηρεί το σύστημα στη βάση δεδομένων. Με άλλα λόγια, το backend διατηρεί σωστή μαθηματική σημασιολογία, ενώ το frontend λαμβάνει «ανθρωποκεντρική» αναπαράσταση.

Επιπλέον, στη δημόσια διεπαφή καλείται συστηματικά η expire_holds() από το reservations/services.py πριν την παρουσίαση δεδομένων. Η εφαρμογή αντιμετωπίζει τα holds ως ενεργά στοιχεία που επηρεάζουν τη διαθεσιμότητα και, συνεπώς, πρέπει να απομακρύνονται όταν λήγουν ώστε να μην εμφανίζονται ψευδώς δεσμευμένες ημέρες. Σε επίπεδο εμπειρίας χρήστη, αυτό αποτρέπει την εμφάνιση «κολλημένων» μη διαθέσιμων ημερομηνιών, ενώ σε επίπεδο συνέπειας, μειώνει την πιθανότητα να επηρεάζουν ληγμένες κρατήσεις την ορατή διαθεσιμότητα.

6.4 Μηχανισμός Κρατήσεων και Διαθεσιμότητας

Ο μηχανισμός κρατήσεων αποτελεί το κεντρικό υποσύστημα, καθώς υλοποιεί τον αλγόριθμο που καθορίζει εάν ένα αίτημα μπορεί να γίνει αποδεκτό, πώς δεσμεύονται προσωρινά ημερομηνίες, και πώς μεταβαίνει μια κράτηση από εκκρεμή σε τελική κατάσταση. Στο έργο, ο πυρήνας αυτής της λογικής βρίσκεται στο reservations/services.py, ενώ τα views λειτουργούν ως «ελεγκτές εισόδου» που συλλέγουν δεδομένα από HTTP requests, τα καθαρίζουν και τα μεταβιβάζουν στις υπηρεσίες.

Η σχεδίαση αυτή επιτρέπει να διατηρηθεί μια συνεκτική και ενιαία πολιτική ελέγχων. Η διαθεσιμότητα ελέγχεται με τον ίδιο τρόπο είτε η κράτηση προέρχεται από δημόσιο χρήστη είτε

εγκρίνεται από τον ιδιοκτήτη, καθώς και στις δύο περιπτώσεις ο έλεγχος βασίζεται στην ίδια συνάρτηση (is_range_available).

Με αυτόν τον τρόπο, αποφεύγεται το σφάλμα να υλοποιηθούν οι κανόνες παραπάνω από 1 φορές, το οποίο οδηγεί σε αποκλίσεις και σφάλματα συντήρησης.

Τέλος, η υλοποίηση περιλαμβάνει αντιμετώπιση των «ενδιάμεσων» καταστάσεων όπου η κράτηση δεν έχει εγκριθεί αλλά δεσμεύει ημερομηνίες προσωρινά. Η έννοια αυτή ενσωματώνεται στο μοντέλο Booking με το πεδίο hold_expires_at και με τη μέθοδο is_hold_active(). Έτσι, το σύστημα μπορεί να αναγνωρίζει πότε ένα εκκρεμές αίτημα πρέπει να επηρεάζει τη διαθεσιμότητα και πότε όχι, χωρίς να απαιτείται ξεχωριστός πίνακας ή άλλα flags.

6.4.1 Έλεγχος Διαθεσιμότητας

Ο έλεγχος διαθεσιμότητας υλοποιείται στη συνάρτηση is_range_available(accommodation, check_in, check_out, ignore_booking_id=None). Η συνάρτηση εφαρμόζει πρώτα έλεγχο εγκυρότητας του διαστήματος (π.χ. check_in >= check_out οδηγεί σε άμεση απόρριψη), και στη συνέχεια συνθέτει ένα ερώτημα που εξετάζει επικαλύψεις με κρατήσεις. Η επικάλυψη υπολογίζεται με κριτήριο check_in < other.check_out AND other.check_in < check_out (Allen, 1983).

Η σύνθεση του ερωτήματος γίνεται με Q αντικείμενα και με διακριτή πολιτική ως προς το ποιες κρατήσεις θεωρούνται «δεσμευτικές». Συγκεκριμένα, περιλαμβάνονται πάντα οι ΕΠΙΒΕΒΑΙΩΜΕΝΕΣ κρατήσεις, ενώ για τα εκκρεμή αιτήματα εισάγεται ο μηχανισμός _effective_pending_filter(), ο οποίος λαμβάνει υπόψη τα ενεργά holds. Έτσι, ο έλεγχος δεν θεωρεί όλα τα ΣΕ ΕΚΚΡΕΜΟΤΗΤΑ αιτήματα δεσμευτικά, αλλά μόνο όσα έχουν hold_expires_at στο μέλλον. Η διαφοροποίηση αυτή επιτρέπει στο σύστημα να χρησιμοποιεί το ΣΕ ΕΚΚΡΕΜΟΤΗΤΑ ως πραγματική επιχειρησιακή κατάσταση (αναμονή έγκρισης) χωρίς να εγκλωβίζει μόνιμα τη διαθεσιμότητα.

Πέρα από κρατήσεις, ο έλεγχος ενσωματώνει και τις περιόδους μη διαθεσιμότητας (BlockedPeriod). Το ερώτημα που εφαρμόζεται στα blocks χρησιμοποιεί την ίδια λογική επικάλυψης, δηλαδή start_date__lt=check_out και end_date__gt=check_in. Με αυτό τον τρόπο,

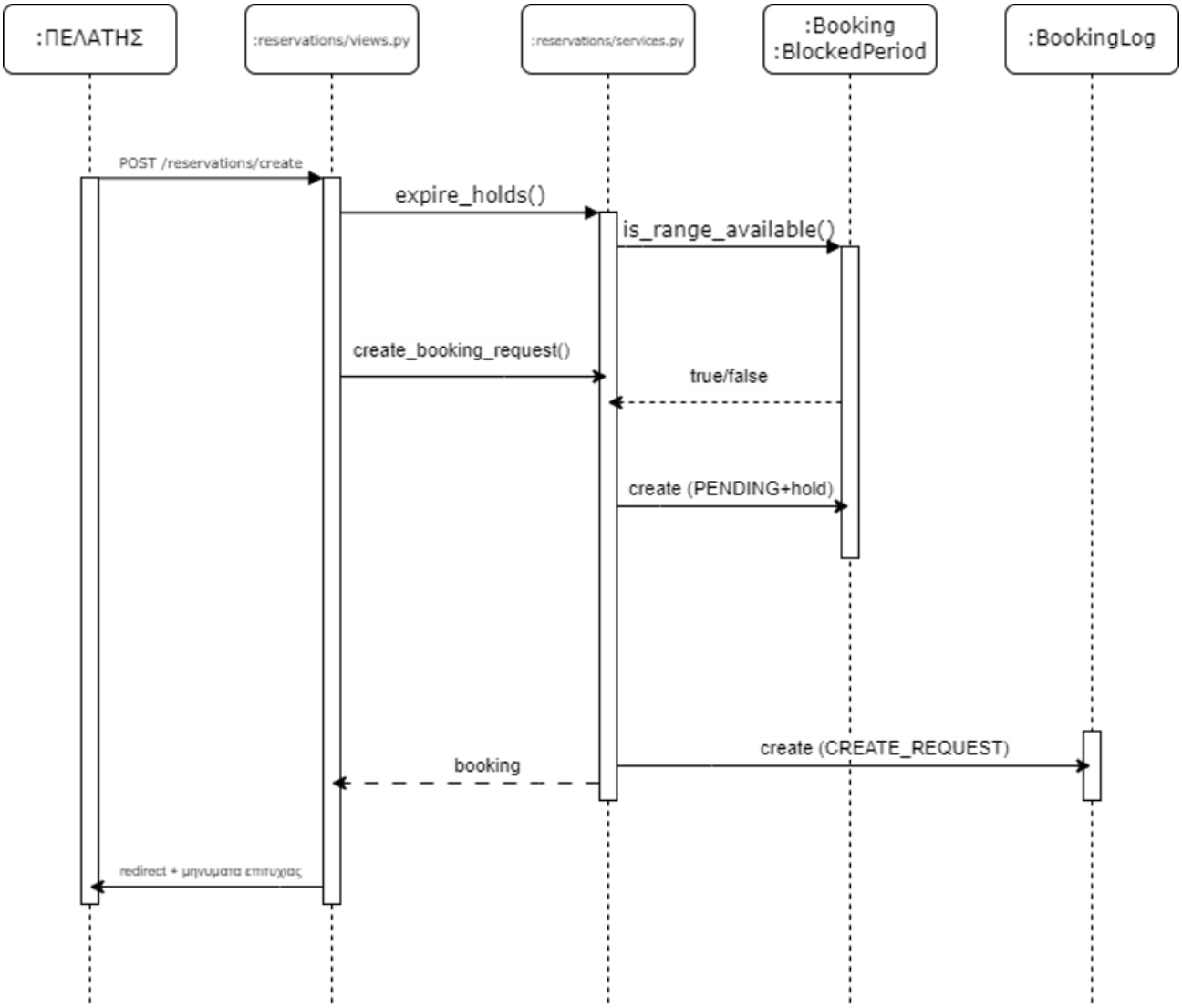
το σύστημα χειρίζεται ομοιόμορφα διαφορετικούς τύπους «δεσμεύσεων», εξασφαλίζοντας ότι δεν μπορεί να εγκριθεί κράτηση σε διάστημα συντήρησης ή ιδιοχρησίας. Το αποτέλεσμα είναι ένας ενιαίος, επαναχρησιμοποιήσιμος έλεγχος που συνδυάζει δεδομένα από πολλαπλές οντότητες χωρίς να απαιτείται περίπλοκη λογική στο επίπεδο των views.

6.4.2 Δημιουργία Αιτήματος Κράτησης

Η δημιουργία αιτήματος κράτησης πραγματοποιείται στο reservations/views.py μέσω endpoint που αναμένει POST δεδομένα. Το view εφαρμόζει αρχικά προστατευτικά φίλτρα, εάν το αίτημα δεν είναι POST, γίνεται ανακατεύθυνση στην αρχική σελίδα, αποτρέποντας μη αναμενόμενες κλήσεις. Στη συνέχεια, εκτελείται expire_holds(), ώστε η διαδικασία να βασιστεί σε «καθαρή» εικόνα διαθεσιμότητας. Κατόπιν, ανακτώνται και καθαρίζονται τα πεδία first_name, last_name, country, email, phone και customer_notes, ενώ οι ημερομηνίες μετατρέπονται από ISO string σε date μέσω βοηθητικής συνάρτησης _parse_iso_date που αξιοποιεί parse_date. Η κρίσιμη μεταφορά της λογικής γίνεται όταν το view καλεί create_booking_request(...) από το reservations/services.py. Εκεί, πριν δημιουργηθεί εγγραφή, πραγματοποιείται έλεγχος ορίων φιλοξενούμενων με βάση το accommodation.max_guests, καθώς και έλεγχος ότι check_in < check_out. Το σημείο αυτό αποτυπώνει ότι η εγκυρότητα δεν εξαρτάται μόνο από τον χρήστη/φόρμα, αλλά επιβεβαιώνεται και στο επίπεδο υπηρεσίας, γεγονός που ενισχύει την ανθεκτικότητα σε σφάλματα ή κακόβουλες κλήσεις.

Η δημιουργία της κράτησης υλοποιείται με Booking.objects.create(...) και περιλαμβάνει «snapshot» πεδία πελάτη (όνομα, επώνυμο, χώρα, email, τηλέφωνο). Η επιλογή snapshot, η οποία δηλώνεται ρητά στο μοντέλο ως “Customer snapshot fields”, διασφαλίζει ότι η κράτηση παραμένει ιστορικά σταθερή ακόμη και αν τα στοιχεία λογαριασμού αλλάξουν. Παράλληλα, η κράτηση ξεκινά ως ΣΕ ΕΚΚΡΕΜΟΤΗΤΑ και αποκτά hold_expires_at μέσω _hold_expires_at(), που βασίζεται σε χρονική διάρκεια (settings.BOOKING_HOLD_MINUTES). Με αυτόν τον τρόπο, το σύστημα υλοποιεί ένα προσωρινό «παράθυρο» κράτησης όπου ο πελάτης δεσμεύει ημερομηνίες, αλλά η δέσμευση λήγει εάν ο ιδιοκτήτης δεν αποφασίσει εγκαίρως ή εάν το σύστημα επιλέγει να καθαρίσει παλαιά holds. Επιπλέον, μετά τη δημιουργία, καταγράφεται ενέργεια στο BookingLog με τύπο CREATE_REQUEST, ενισχύοντας την ιχνηλασιμότητα της ροής.

Η πλήρης αλληλουχία των κλήσεων κατά τη δημιουργία ενός αιτήματος κράτησης παρουσιάζεται στο διάγραμμα ακολουθίας της Εικόνας 8. Το διάγραμμα καθιστά εμφανή τον διαχωρισμό ευθυνών μεταξύ view και υπηρεσίας, τη θέση του ελέγχου διαθεσιμότητας εντός της συναλλαγής, καθώς και την καταγραφή της ενέργειας στο BookingLog πριν από την επιστροφή της απόκρισης στον χρήστη.



Εικόνα 8 Διάγραμμα ακολουθίας για τη δημιουργία αιτήματος κράτησης

6.5 Διαχείριση Κρατήσεων από Ιδιοκτήτη

Η διαχείριση κρατήσεων από τον ιδιοκτήτη υλοποιείται στο dashboard/views.py και διαχωρίζεται σαφώς από τη δημόσια διεπαφή. Η πρόσβαση στις διαχειριστικές λειτουργίες δεν επιτυγχάνεται μόνο με login_required, αλλά με ειδικό μηχανισμό ελέγχου ρόλου, υλοποιημένο μέσω decorator _owner_required. Στον κώδικα, ο decorator αυτός λειτουργεί ως «πύλη» που επιβάλλει ότι οι κρίσιμες ενέργειες (έγκριση/απόρριψη, δημιουργία blocks, απόφαση σε αιτήματα αλλαγών) μπορούν να εκτελεστούν αποκλειστικά από χρήστη που αναγνωρίζεται ως ιδιοκτήτης. Η επιλογή αυτή δεν είναι απλώς θέμα ασφάλειας, αλλά και σαφούς ορισμού επιχειρησιακής αρμοδιότητας στο σύστημα.

Στη διαχειριστική ροή, το σύστημα παρέχει λεπτομέρειες κράτησης μέσω της dashboard_booking_detail, η οποία φορτώνει οποιαδήποτε κράτηση με get_object_or_404 και την αποδίδει σε template του dashboard. Το σημείο αυτό υπογραμμίζει ότι στο dashboard η ορατότητα είναι ευρύτερη σε σχέση με τον πελάτη, ο οποίος (στο reservations/booking_detail) περιορίζεται αυστηρά στις κρατήσεις που ανήκουν στον ίδιο. Έτσι, η διαφοροποίηση «ιδιοκτήτης» και «πελάτης» δεν προκύπτει από το UI, αλλά επιβάλλεται σε επίπεδο queries.

Η έγκριση και απόρριψη κρατήσεων υλοποιούνται με POST endpoints (approve_booking_view, reject_booking_view) που καλούν αντίστοιχα approve_booking και reject_booking από τα services. Η χρήση POST ως απαίτηση για μεταβολές κατάστασης συνάδει με ορθές πρακτικές HTTP (αποφυγή side effects μέσω GET). Επιπλέον, η επικοινωνία αποτελέσματος προς τον χρήστη πραγματοποιείται με το Django messages framework (messages.success, messages.error), στοιχείο που ενισχύει τη χρηστικότητα αλλά και προσφέρει σαφή ανατροφοδότηση για αποτυχίες (π.χ. έγκριση που αποτυγχάνει λόγω μη διαθεσιμότητας).

6.6 Ημερολόγιο και API Διαθεσιμότητας

Η ημερολογιακή απεικόνιση διαθεσιμότητας αποτελεί απαιτητικό στοιχείο, διότι προϋποθέτει μετατροπή ετερογενών πηγών «δέσμευσης» (εγκεκριμένες κρατήσεις, ενεργά holds, blocks) σε ένα κοινό μοντέλο γεγονότων κατάλληλο για παρουσίαση. Το έργο υλοποιεί αυτό το επίπεδο

αφαίρεσης μέσω του AvailabilityEvent (dataclass) και της συνάρτησης get_events_for_range στο reservations/services.py.

Η χρήση dataclass επιτρέπει να επιστρέφεται ένα τυποποιημένο αντικείμενο γεγονότος με πεδία όπως τύπος, χρονικό διάστημα και ενδεχόμενη περιγραφή, διευκολύνοντας τόσο την API έξοδο όσο και την περαιτέρω επεξεργασία.

Το API endpoint στο reservations/views.py (ενδεικτικά availability_json) παράγει JSON απάντηση μετά από parsing ημερομηνιών και επιλογή καταλύματος. Η αρχιτεκτονική σημασία του API είναι ότι διαχωρίζει πλήρως την «πηγή αλήθειας» από τον τρόπο απεικόνισης. Με άλλα λόγια, ο υπολογισμός διαθεσιμότητας παραμένει στο backend και σε μια ενιαία υπηρεσία, ενώ το frontend (templates ή πιθανή μελλοντική JS διεπαφή) καταναλώνει δομημένα δεδομένα γεγονότων. Αυτό καθιστά την εφαρμογή ανθεκτική σε αλλαγές παρουσίασης, όπως διαφορετικό ημερολογιακό widget ή προσθήκη φίλτρων ανά κατάλυμα, χωρίς να απαιτείται τροποποίηση του επιχειρησιακού πυρήνα.

6.7 Έλεγχος Πρόσβασης και Ρόλων Χρηστών

Ο έλεγχος πρόσβασης στο έργο υλοποιείται σε δύο επίπεδα, στο επίπεδο views (με decorators) και στο επίπεδο επιχειρησιακής λογικής (μέσω ελέγχων κατάστασης/δικαιωμάτων στις υπηρεσίες). Για τον πελάτη, η πρόσβαση σε λειτουργίες όπως “my bookings” ή “booking detail” επιβάλλεται με login_required. Στη booking_detail, ο κώδικας χρησιμοποιεί get_object_or_404(Booking, id=booking_id, customer=request.user), πράγμα που σημαίνει ότι ακόμη και αν ο χρήστης γνωρίζει το booking_id, δεν μπορεί να δει κράτηση άλλου πελάτη. Πρόκειται για μορφή authorization μέσω φίλτρου query, η οποία είναι ισχυρότερη από επιφανειακούς ελέγχους σε template.

Για τον ιδιοκτήτη, ο έλεγχος είναι αυστηρότερος και πιο εξειδικευμένος μέσω _owner_required. Το decorator υπάρχει καθώς η αυθεντικοποίηση (login) δεν επαρκεί, διότι απαιτείται διαφοροποίηση ρόλων. Η διαφοροποίηση ενισχύεται και από την επιλογή να τοποθετηθούν οι διαχειριστικές λειτουργίες σε ξεχωριστό app (dashboard), ώστε οι ρυθμίσεις πρόσβασης να είναι

εγγενές χαρακτηριστικό του υποσυστήματος και όχι «πρόσθετη συνθήκη» διάσπαρτη στον κώδικα.

Το δεύτερο επίπεδο προστασίας προκύπτει από τη διαχείριση καταστάσεων στις υπηρεσίες. Για παράδειγμα, στην reject_booking εφαρμόζεται ρητά ο κανόνας ότι «μόνο pending bookings μπορούν να απορριφθούν». Ομοίως, στην έγκριση εφαρμόζεται ο κανόνας ότι μια κράτηση δεν μπορεί να εγκριθεί εάν το εύρος της δεν είναι διαθέσιμο κατά τον χρόνο απόφασης. Έτσι, ακόμη και αν ένα view κληθεί κατά λάθος ή σε μη αναμενόμενο σενάριο, η υπηρεσία θα αποτρέψει λογικά άκυρη μετάβαση.

6.9 Συνολική Αξιολόγηση Υλοποίησης

Η υλοποίηση του συστήματος, συγκροτεί ένα συνεκτικό και λειτουργικό παράδειγμα εφαρμογής αρχών μηχανικής λογισμικού σε πραγματικό επιχειρησιακό αντικείμενο. Το ισχυρότερο χαρακτηριστικό είναι ότι οι κρίσιμες αποφάσεις, διαθεσιμότητα, μεταβάσεις καταστάσεων, προσωρινές δεσμεύσεις κτλπ, δεν διαχειρίζονται αποσπασματικά, αλλά ενοποιούνται σε υπηρεσίες με καθαρά όρια και επαναχρησιμοποίηση. Η χρήση transaction.atomic και select_for_update στις αποφάσεις ιδιοκτήτη, η καθολική εφαρμογή του [check_in, check_out) και η ενσωμάτωση των holds μέσω hold_expires_at συγκροτούν μια υλοποίηση που στοχεύει στη συνέπεια υπό πραγματικές συνθήκες χρήσης (Django Software Foundation, 2024).

Επιπλέον, το σύστημα ενσωματώνει ιστορικότητα μέσω BookingLog και ActionType, ώστε οι κρίσιμες ενέργειες να καταγράφονται. Η πρακτική αυτή συνδέεται με τις απαιτήσεις αξιοπιστίας και διαχειρισιμότητας αφού ο ιδιοκτήτης μπορεί να αιτιολογεί αποφάσεις, το σύστημα μπορεί να ελεγχθεί εκ των υστέρων, και οι μεταβολές δεν αποτελούν «σιωπηρές» αλλαγές στη βάση. Η ιστορικότητα, σε συνδυασμό με το dashboard, μετατρέπει την εφαρμογή από μια απλή φόρμα κράτησης σε διαχειριστικό εργαλείο με οργανωμένη ροή.

Συνολικά, η υλοποίηση επιτυγχάνει τον στόχο της ανάλυσης για την υλοποίηση ενός σύστηματος κρατήσεων που δεν είναι μόνο λειτουργικό ως προς την υποβολή αιτημάτων, αλλά και στιβαρό ως προς τη λογική διαθεσιμότητας, την ασφάλεια πρόσβασης και τη διαχείριση αποφάσεων. Η δομή και οι μηχανισμοί του έργου επιτρέπουν επίσης φυσική συνέχεια στα επόμενα κεφάλαια (UI/UX, δοκιμές, αξιολόγηση), διότι παρέχουν σαφή σημεία ελέγχου και μετρήσιμη συμπεριφορά για σενάρια χρήσης.

ΚΕΦΑΛΑΙΟ 7ο

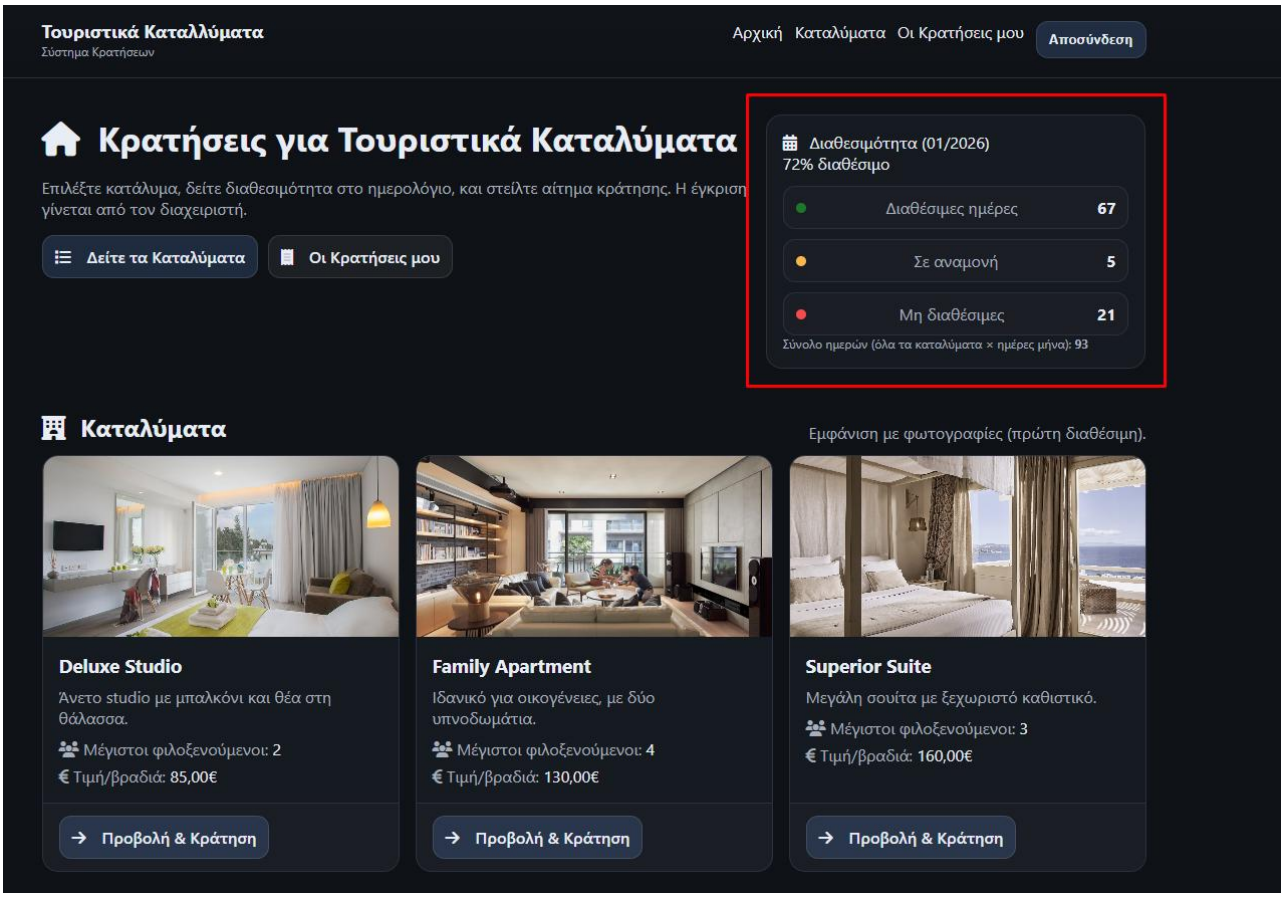
7. ΔΙΕΠΑΦΗ ΧΡΗΣΤΗ ΚΑΙ ΕΜΠΕΙΡΙΑ ΧΡΗΣΤΗ (UI/UX)

7.1 Σχεδίαση και Αρχές UI/UX

Η διεπαφή χρήστη του συστήματος κρατήσεων σχεδιάστηκε με κεντρικό άξονα τη σαφήνεια της πληροφορίας και τη μείωση του γνωστικού φορτίου κατά την αλληλεπίδραση. Σε εφαρμογές τουριστικών κρατήσεων, η πληροφορία που καθορίζει την επιτυχία της εμπειρίας είναι πρωτίστως χρονική, ο χρήστης ενδιαφέρεται για το «πότε» μπορεί να διαμείνει και για το «αν» το επιλεγμένο διάστημα είναι διαθέσιμο. Για τον λόγο αυτό, ο πυρήνας του UI/UX συνδέεται άμεσα με τον μηχανισμό διαθεσιμότητας που παράγεται στο backend και καταναλώνεται από τα templates.Η προτεραιότητα αυτή αποτυπώνεται με το ότι η εφαρμογή υπολογίζει και παρέχει στην αρχική σελίδα συγκεντρωτικά μεγέθη διαθεσιμότητας ανά μήνα, μέσω του booking_site/core/views.py και του context που αποδίδεται στο template core/home.html. Συγκεκριμένα, στο context μεταβιβάζονται τα month_label και οι μετρικές availability_total_days, availability_available_days, availability_pending_days, availability_unavailable_days και availability_pct, οι οποίες επιτρέπουν στη διεπαφή να παρουσιάζει άμεσα μια συνοπτική «εικόνα» κατάστασης χωρίς ο χρήστης να χρειάζεται να πλοηγηθεί βαθιά στο σύστημα (ISO/IEC, 2011).

Η σχεδίαση ακολουθεί επίσης την αρχή της συνεχούς ανατροφοδότησης (feedback) και της αποφυγής «σιωπηρών αποτυχιών». Αυτό υλοποιείται καθολικά μέσω του Django messages framework, το οποίο χρησιμοποιείται τόσο στη δημόσια πλευρά όσο και στο dashboard. Ενδεικτικά, στο reservations/views.py το endpoint create_booking_request_view εμφανίζει μήνυμα σφάλματος όταν δεν έχουν δοθεί έγκυρες ημερομηνίες («Παρακαλώ επέλεξε έγκυρες ημερομηνίες.») και, αντίστοιχα, μήνυμα επιτυχίας όταν το αίτημα καταχωρείται («Το αίτημα κράτησης καταχωρήθηκε και αναμένει έγκριση.»). Η επιλογή αυτή μεταφέρει στον χρήστη την

ακριβή κατάσταση της ενέργειάς του και μειώνει την αβεβαιότητα, ειδικά σε ροές όπου η κράτηση δεν ολοκληρώνεται άμεσα αλλά μπαίνει σε διαδικασία έγκρισης (Nielsen, 2024).



Εικόνα 9 Εμφάνιση στατιστικών στοιχείων στην αρχική οθόνη

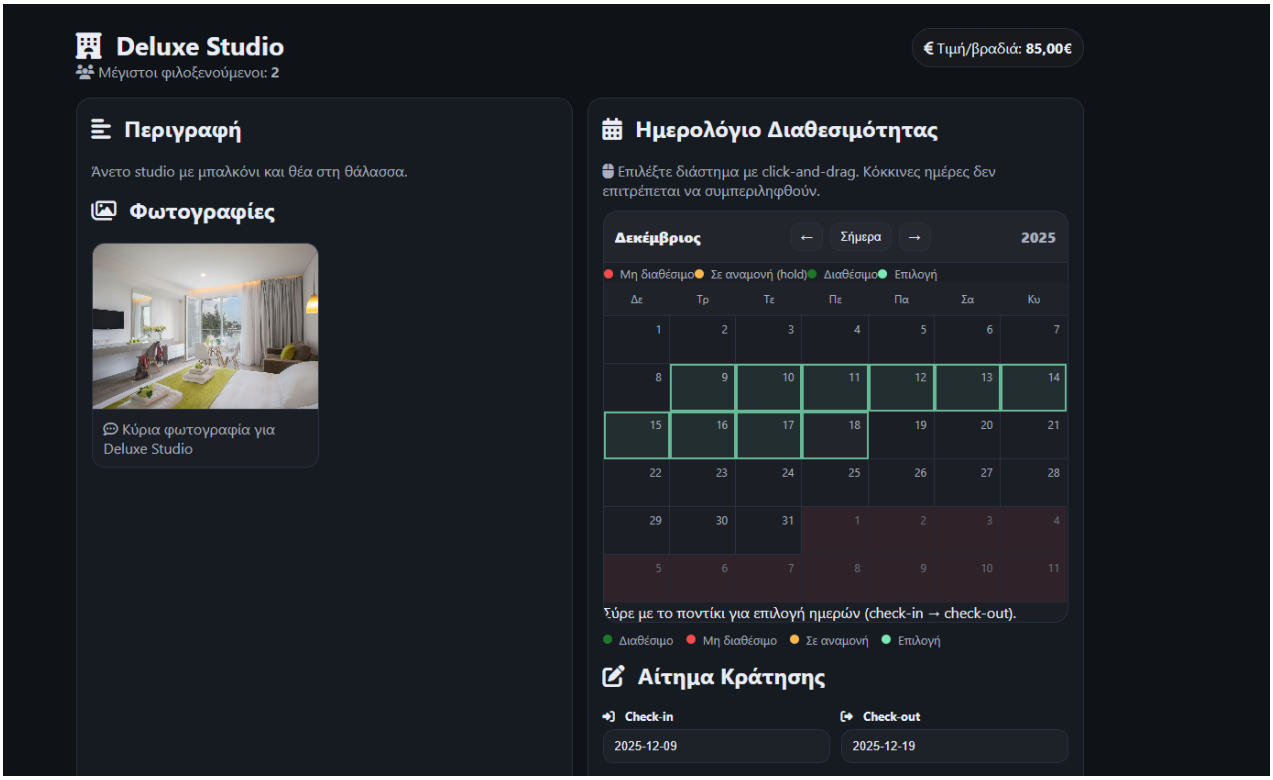
Ένα επιπλέον σχεδιαστικό χαρακτηριστικό που επηρεάζει την εμπειρία χρήστη είναι η συνέπεια της «πηγής αλήθειας» για τη διαθεσιμότητα και ο τρόπος που αυτή επικαιροποιείται. Σε πολλά συστήματα εμφανίζονται φαινόμενα όπου παλαιά αιτήματα συνεχίζουν να δεσμεύουν ημέρες και να προκαλούν σύγχυση στη διεπαφή. Στον κώδικα του έργου, το πρόβλημα αντιμετωπίζεται με την συστηματική κλήση της expire_holds() πριν από κρίσιμες σελίδες ή λίστες, τόσο στο δημόσιο τμήμα όσο και στο dashboard (π.χ. στην create_booking_request_view, στη my_bookings, αλλά και στη dashboard_index). Η τεχνική αυτή έχει άμεση UI/UX συνέπεια αφού ο χρήστης δεν βλέπει «κολλημένες» δεσμεύσεις που δεν αντανακλούν την πραγματική λειτουργική κατάσταση, ενώ ο ιδιοκτήτης διαχειρίζεται μόνο ενεργά αιτήματα που έχουν ουσιαστική σημασία.

7.2 Ροή Χρήστη για Υποβολή Κράτησης

Η ροή υποβολής κράτησης έχει σχεδιαστεί ως μια διαδικασία που μειώνει τη τριβή αλλά ταυτόχρονα επιβάλλει τους απαραίτητους κανόνες εγκυρότητας. Στο επίπεδο της διεπαφής, ο χρήστης ξεκινά από την προβολή του καταλύματος και προχωρά στην επιλογή ημερομηνιών και συμπλήρωση στοιχείων. Στο επίπεδο του backend, αυτό αντιστοιχεί σε μια στοχευμένη μεταφορά δεδομένων μέσω POST προς το reservations/views.py:create_booking_request_view. Η ίδια η συνάρτηση τεκμηριώνει ρητά τα αναμενόμενα πεδία (accommodation_id, check_in, check_out, guests_count, καθώς και στοιχεία πελάτη).

Η εμπειρία χρήστη ενισχύεται από το ότι το σύστημα χειρίζεται άμεσα συχνά λάθη εισόδου χωρίς να «σπάει» η πλοήγηση. Στη create_booking_request_view οι ημερομηνίες αναλύονται από ISO μορφή με την _parse_iso_date (η οποία χρησιμοποιεί parse_date), ενώ για το πλήθος ατόμων γίνεται ασφαλής μετατροπή (try: guests_count = int(...) except ValueError: guests_count = 1). Αυτές οι λεπτομέρειες υπηρετούν την UX αρχή της ανοχής σε σφάλματα. Εάν, για παράδειγμα, ο χρήστης ή το UI στείλει μη έγκυρη τιμή στο guests_count, το σύστημα δεν καταρρέει, αλλά επιλέγει μια προεπιλογή. Αντίστοιχα, όταν δεν υπάρχουν έγκυρες ημερομηνίες (if not check_in or not check_out:), εμφανίζεται μήνυμα σφάλματος και ο χρήστης επιστρέφει στη σελίδα του καταλύματος (redirect("accommodations:detail", ...)) ώστε να διορθώσει την επιλογή του. Η επιλογή επιστροφής στο κατάλληλο context (τη σελίδα του συγκεκριμένου καταλύματος) είναι ουσιώδης για τη ροή, διότι ο χρήστης δεν «χάνεται» σε μια γενική σελίδα αλλά επιστρέφει στο σημείο όπου έχει νόημα να επαναλάβει την ενέργεια.

Σημαντικό στοιχείο της ροής είναι ότι η εφαρμογή αποτυπώνει τη λογική «αίτημα και έγκριση» ως κεντρικό UX σενάριο. Μετά την επιτυχή δημιουργία του αιτήματος μέσω create_booking_request(...), ο χρήστης ενημερώνεται ότι η ενέργεια καταχωρήθηκε και αναμένει έγκριση. Δεν υπονοείται άμεση δέσμευση ως οριστική κράτηση, αλλά υπάρχει σαφές στάδιο απόφασης. Επιπλέον, η συνέχεια της πλοήγησης προσαρμόζεται στην κατάσταση αυθεντικοποίησης, δηλαδή αν ο χρήστης είναι συνδεδεμένος, γίνεται ανακατεύθυνση σε λεπτομέρειες κράτησης (redirect("reservations:booking_detail", booking_id=booking.id)), ενώ αν είναι ανώνυμος επιστρέφει στο κατάλυμα.

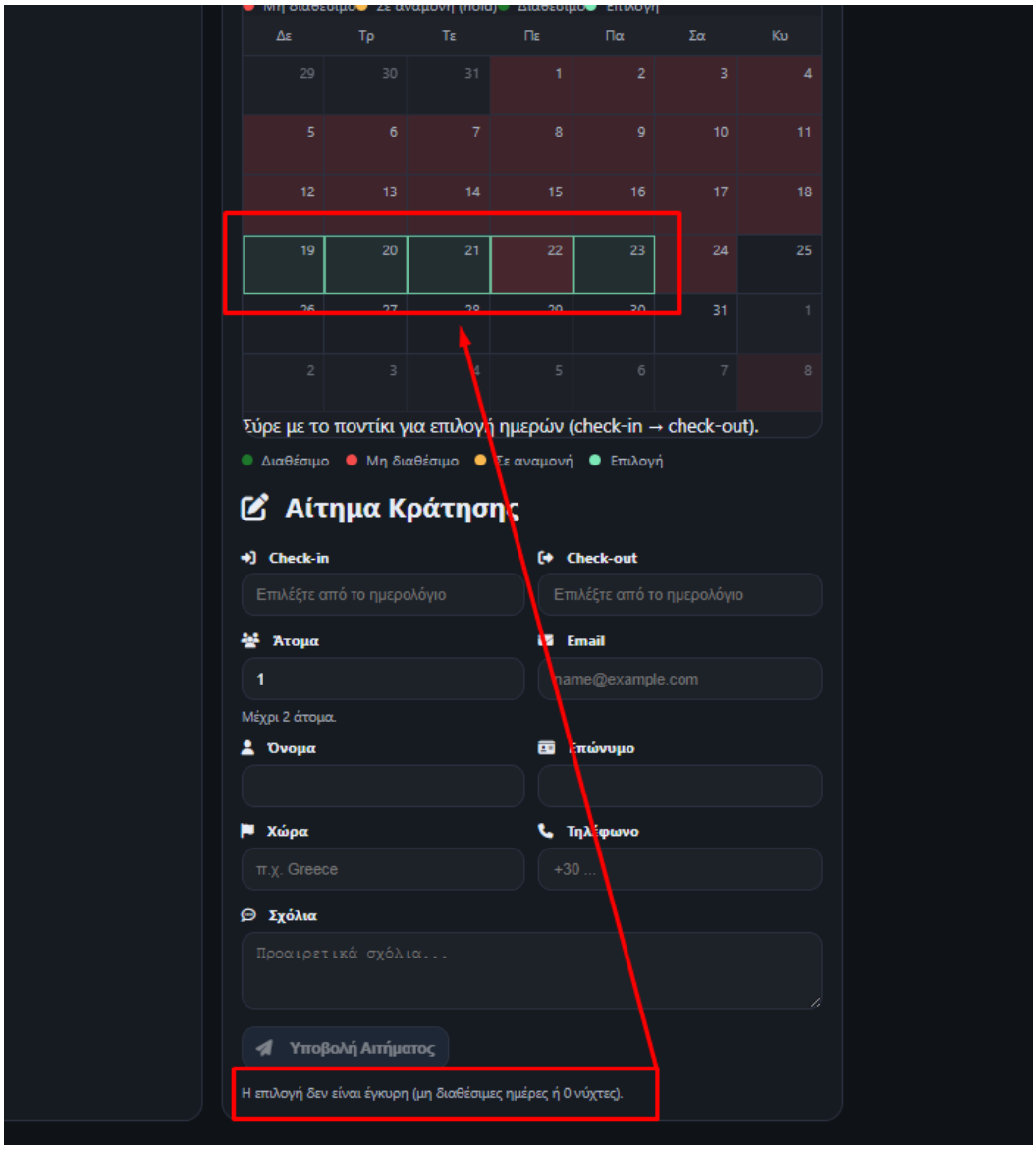


Εικόνα 10 Έγκυρη επιλογή κράτησης

Η διάκριση αυτή έχει σαφή UX λογική. Ο συνδεδεμένος χρήστης αναμένει «προσωπικό χώρο» όπου μπορεί να δει την κατάσταση του αιτήματος, ενώ ο ανώνυμος χρήστης εξυπηρετείται καλύτερα με την επιστροφή στο περιεχόμενο που ήδη παρακολουθούσε. Η αντίστοιχη σελίδα λίστας κρατήσεων (my_bookings) υλοποιείται με Booking.objects.filter(customer=request.user).order_by("-created_at"), δηλαδή εμφανίζει χρονικά τις ενέργειες του χρήστη, κάτι που αποτελεί κοινή και κατανοητή πρακτική διεπαφής για ιστορικό ενεργειών.

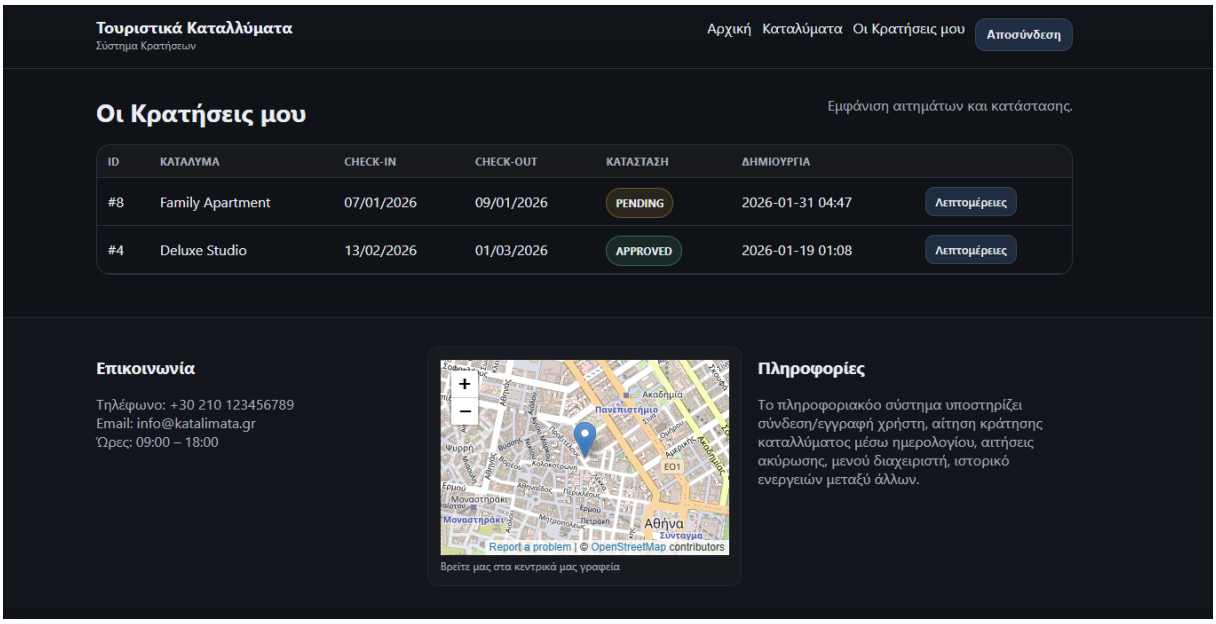
Επιπλέον, η ροή χρήστη επεκτείνεται και μετά την αρχική υποβολή, καθώς ο χρήστης μπορεί να αιτηθεί ακύρωση ή τροποποίηση. Αυτό υποστηρίζεται από τα endpoints request_cancellation_view και request_modification_view στο reservations/views.py. Η σχεδίαση εδώ είναι σημαντική από πλευράς UX καθώς το σύστημα δεν αντιμετωπίζει την κράτηση ως «τετελεσμένο γεγονός», αλλά ως οντότητα με κύκλο ζωής και δυνατότητα αλλαγών. Αυτό δημιουργεί αίσθηση ελέγχου και εμπιστοσύνης, ιδίως σε περιβάλλοντα όπου οι ημερομηνίες ταξιδιού μεταβάλλονται συχνά. Η ύπαρξη τέτοιων endpoints προϋποθέτει

αντίστοιχες σελίδες/φόρμες στο UI, αλλά κυρίως αποτυπώνει ότι το UX δεν σταματά στη στιγμή της υποβολής αλλά συνεχίζει με μηχανισμούς διαχείρισης αιτημάτων από τον πελάτη.



Εικόνα 11 Μη έγκυρη επιλογή κράτησης (Μη διαθέσιμες ημέρες απο άλλη κράτηση έχουν επιλεχθεί)

Η Εικόνα 12 παρουσιάζει τη σελίδα «Οι κρατήσεις μου», όπου ο συνδεδεμένος πελάτης βλέπει τις κρατήσεις του με την τρέχουσα κατάστασή τους, σε φθίνουσα χρονολογική σειρά δημιουργίας.



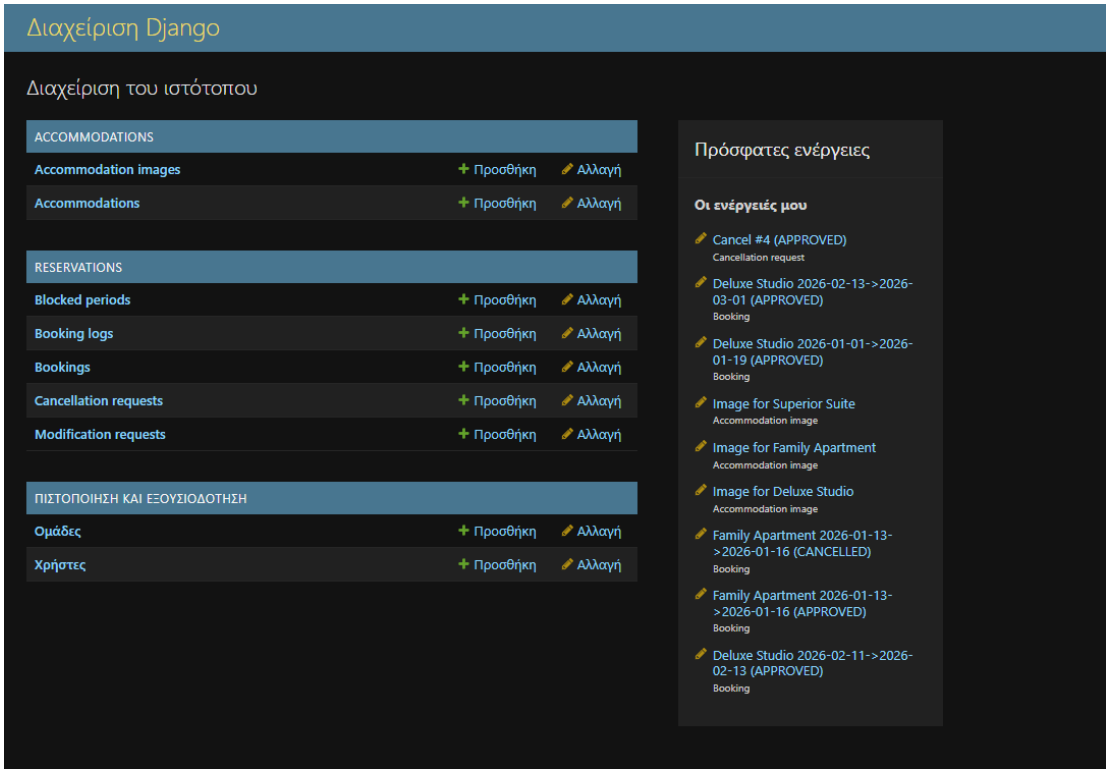
Εικόνα 12 Μενού - Οι κρατήσεις μου (Διαθέσιμο για συνδεδεμένους χρήστης)

7.3 Πίνακας Διαχείρισης Ιδιοκτήτη

Ο πίνακας διαχείρισης (dashboard) σχεδιάστηκε ως ξεχωριστή εμπειρία χρήστη, με διαφορετικές ανάγκες από τη δημόσια διεπαφή. Ενώ ο πελάτης απαιτεί καθοδήγηση και απλότητα, ο ιδιοκτήτης απαιτεί εποπτεία, συγκέντρωση εκκρεμοτήτων και δυνατότητα λήψης αποφάσεων με σαφή τεκμηρίωση. Η υλοποίηση στο dashboard/views.py αποτυπώνει αυτή τη λογική ξεκινώντας από τον μηχανισμό πρόσβασης, οι διαχειριστικές λειτουργίες προστατεύονται από τον decorator _owner_required, ο οποίος βασίζεται σε έλεγχο ρόλου (_is_owner). Η προσέγγιση αυτή διαχωρίζει ρητά το UX των ρόλων και αποτρέπει να εμφανίζονται λειτουργίες υψηλού κινδύνου (όπως η έγκριση/απόρριψη) σε μη εξουσιοδοτημένους χρήστες.

Το κεντρικό view του dashboard (dashboard_index) συγκροτεί την εμπειρία γύρω από την αρχή της προτεραιοποίησης εργασιών. Στον κώδικα, ανακτώνται οι εκκρεμείς κρατήσεις με Booking.objects.filter(status=BookingStatus.PENDING).order_by("created_at")[:50], καθώς και εκκρεμή αιτήματα ακύρωσης και τροποποίησης από τις αντίστοιχες οντότητες (CancellationRequest, ModificationRequest). Η επιλογή ορίου (μέσω [:50]) και η ταξινόμηση κατά created_at περιλαμβάνουν μια UX υπόθεση ότι ο ιδιοκτήτης πρέπει να βλέπει πρώτα τα παλαιότερα εκκρεμή, ώστε να μειώνεται ο χρόνος αναμονής. Παράλληλα, το context

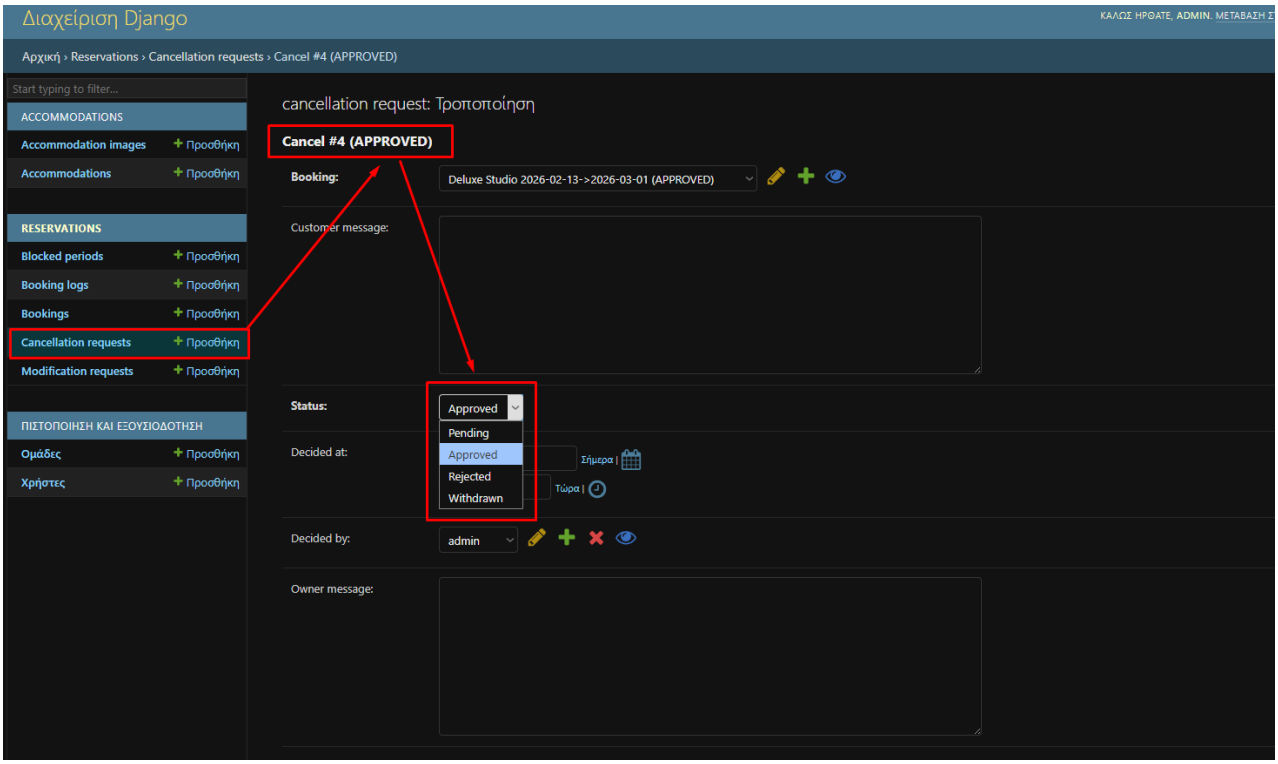
περιλαμβάνει και την λίστα ενεργών καταλυμάτων, επιτρέποντας στο UI να παρέχει γρήγορη πρόσβαση σε φίλτρα ή σε επιλογές συσχέτισης ενεργειών με συγκεκριμένο κατάλυμα. Η ενσωμάτωση αυτών των στοιχείων στο αρχικό dashboard μειώνει τον αριθμό βημάτων που απαιτούνται για την εκτέλεση της διαχειριστικής εργασίας.



Εικόνα 13 Μενού επιλογών διαχειριστή

Η διαδικασία έγκρισης και απόρριψης ενσωματώνει στοιχεία UX που εξισορροπούν την ταχύτητα με την ιχνηλασιμότητα. Τα endpoints approve_booking_view και reject_booking_view απαιτούν POST, ανακτούν προαιρετικές σημειώσεις ιδιοκτήτη από το πεδίο owner_notes και καλούν τις αντίστοιχες υπηρεσίες (approve_booking, reject_booking). Στη συνέχεια, το σύστημα ενημερώνει τον ιδιοκτήτη με ξεκάθαρα μηνύματα («Η κράτηση εγκρίθηκε.» ή «Η κράτηση απορρίφθηκε.») ή με μήνυμα αποτυχίας που περιλαμβάνει την εξαίρεση. Η πρακτική αυτή έχει διπλή UX σημασία. Πρώτον, παρέχει άμεση επιβεβαίωση ότι η ενέργεια ολοκληρώθηκε. Δεύτερον, όταν αποτυγχάνει, δεν αποκρύπτει το γεγονός, αλλά μεταφέρει πληροφορία για το τι συνέβη, κάτι που είναι χρήσιμο σε περιβάλλον όπου η αποτυχία μπορεί να οφείλεται σε μεταβολή διαθεσιμότητας ή σε περιορισμούς κατάστασης. Η ανακατεύθυνση πίσω στη σελίδα

λεπτομερειών (redirect("dashboard:booking_detail", ...)) κρατά τον ιδιοκτήτη στο σωστό πλαίσιο εργασίας, επιτρέποντας επανέλεγχο των δεδομένων ή επανάληψη της ενέργειας εφόσον διορθωθεί το πρόβλημα.



Εικόνα 14 Προβολή και διαχείριση αιτημάτων μέσα απο την πλατφόρμα του διαχειριστή (Παράδειγμα - Επιβεβαίωση αίτηματος ακύρωσης κράτησης απο πελάτη)

ΚΕΦΑΛΑΙΟ 8ο

8. ΕΛΕΓΧΟΣ ΚΑΙ ΔΟΚΙΜΕΣ

8.1 Στρατηγική Δοκιμών

Ο έλεγχος ενός συστήματος κρατήσεων απαιτεί μεθοδολογία που υπερβαίνει τη γενική επαλήθευση επειδή η ορθότητα του λογισμικού εξαρτάται από χρονικές αλληλεπιδράσεις, από μεταβάσεις καταστάσεων και από κανόνες συνέπειας που πρέπει να εφαρμόζονται ομοιόμορφα σε πολλαπλές ροές (δημόσια διεπαφή, API διαθεσιμότητας, dashboard ιδιοκτήτη). Στον κώδικα του έργου, οι κρίσιμες επιχειρησιακές αποφάσεις συγκεντρώνονται στο reservations/services.py (π.χ. is_range_available, create_booking_request, approve_booking, expire_holds, create_block), ενώ τα views (reservations/views.py, dashboard/views.py, booking_site/core/views.py) αναλαμβάνουν κυρίως την οργάνωση της HTTP ροής, την ανάγνωση παραμέτρων και την παροχή ανατροφοδότησης μέσω μηνυμάτων (messages). Αυτή η διάκριση ευνοεί μια στρατηγική δοκιμών που ξεκινά από unit tests/logic tests στο επίπεδο υπηρεσιών και επεκτείνεται σε integration tests στο επίπεδο views, ώστε να επαληθεύεται όχι μόνο η σωστή απόφαση του πυρήνα, αλλά και η ορθή «σύνδεση» της απόφασης με την εμπειρία χρήστη (Django Software Foundation, 2024).

Μια πρακτικά αποτελεσματική προσέγγιση είναι η ιεράρχηση των δοκιμών σε τρία επίπεδα. Πρώτον, οι δοκιμές επιχειρησιακής λογικής πρέπει να επικεντρώνονται σε συναρτήσεις όπως η _overlaps και η is_range_available, διότι αυτές καθορίζουν τη θεμελιώδη υπόθεση του συστήματος ότι δύο δεσμεύσεις (κρατήσεις ή blocks) δεν μπορούν να επικαλύπτονται στο ίδιο κατάλυμα, υπό τη σημασιολογία διαστημάτων [check_in, check_out). Δεύτερον, οι δοκιμές ροής στο επίπεδο υπηρεσιών πρέπει να επαληθεύουν την ορθότητα μεταβάσεων κατάστασης, όπως συμβαίνει στη approve_booking και reject_booking, όπου μια κράτηση μεταβαίνει από BookingStatus.PENDING σε BookingStatus.APPROVED ή BookingStatus.REJECTED, και αντίστοιχα καταγράφεται ενέργεια στο BookingLog μέσω ActionType. Τρίτον, οι δοκιμές

διεπαφής/ενοποίησης πρέπει να ελέγχουν ότι τα views εφαρμόζουν σωστά τους περιορισμούς (π.χ. request.method == "POST" για μεταβολές κατάστασης), επιστρέφουν ορθά redirects, και εμφανίζουν τα κατάλληλα μηνύματα σε επιτυχία ή αποτυχία, όπως φαίνεται στη create_booking_request_view και στη approve_booking_view.

Ιδιαίτερη πρόβλεψη απαιτείται για τις συναλλαγές και τη συμπεριφορά ταυτόχρονης πρόσβασης, επειδή το σύστημα περιλαμβάνει σημεία όπου γίνεται χρήση transaction.atomic() και κλειδώματα γραμμών μέσω select_for_update() (ιδίως σε αποφάσεις ιδιοκτήτη στις υπηρεσίες) (Bernstein, Hadzilacos & Goodman, 1987).

8.2 Σενάρια και Περιπτώσεις Ελέγχου

Οι περιπτώσεις ελέγχου στο παρόν έργο πρέπει να προκύπτουν απευθείας από τις πραγματικές ροές που υλοποιούνται στον κώδικα και να καλύπτουν τις βασικές κατηγορίες κινδύνου:

- (α) λάθη υπολογισμού διαθεσιμότητας,
- (β) λανθασμένες μεταβάσεις κατάστασης,
- (γ) λανθασμένες δεσμεύσεις (holds) που παραμορφώνουν τη διαθεσιμότητα
- (δ) αλληλεπίδραση ιδιοκτήτη-πελάτη μέσω αιτημάτων αλλαγών.

Σε αυτό το πλαίσιο, ο πρώτος πυλώνας δοκιμών είναι η δημιουργία αιτήματος κράτησης. Το endpoint reservations/views.py:create_booking_request_view αναλύει ημερομηνίες μέσω _parse_iso_date (που βασίζεται σε parse_date) και εφαρμόζει άμεσους ελέγχους για κενές/μη έγκυρες ημερομηνίες, πριν καλέσει reservations/services.py:create_booking_request.

Ένα ουσιώδες test σενάριο είναι η υποβολή αιτήματος με check_in >= check_out, το οποίο πρέπει να οδηγεί σε απόρριψη με μήνυμα σφάλματος, χωρίς να δημιουργείται εγγραφή Booking. Αφενός το view πρέπει να διαχειριστεί σωστά το input και το redirect, αφετέρου η υπηρεσία πρέπει να επιβεβαιώνει εκ νέου την εγκυρότητα του διαστήματος ώστε να μην δημιουργούνται κρατήσεις μέσω απευθείας κλήσης της υπηρεσίας.

Ο δεύτερος πυλώνας δοκιμών αφορά την αποφυγή επικαλύψεων, όπου ο κώδικας υιοθετεί την κλειστή-ανοικτή σημασιολογία [check_in, check_out). Η is_range_available ελέγχει επικαλύψεις με κριτήριο αντίστοιχο του check_in < other.check_out AND other.check_in < check_out, και διακρίνει ποιες κρατήσεις είναι δεσμευτικές, οι ΕΠΙΒΕΒΑΙΩΜΕΝΕΣ πάντα, και οι ΥΠΟ ΕΓΓΡΙΣΗ μόνο εφόσον έχουν ενεργό hold μέσω _effective_pending_filter() και hold_expires_at.

Έτσι, ένα σύνολο tests περιλαμβάνει:

- κράτηση ΕΠΙΒΕΒΑΙΩΜΕΝΗ που επικαλύπτει πλήρως το νέο διάστημα
- κράτηση ΕΠΙΒΕΒΑΙΩΜΕΝΗ που ακουμπά ακριβώς στο check_out (όπου δεν πρέπει να θεωρείται επικάλυψη
- ΥΠΟ ΕΓΓΡΙΣΗ κράτηση με ενεργό hold που πρέπει να συμπεριφέρεται ως προσωρινό «κλείδωμα» διαθεσιμότητας.

Η αξία αυτών των tests είναι ότι επιβεβαιώνουν στην πράξη τη σημασιολογία του χρόνου που επέλεξε το σύστημα για την αποφυγή διπλοκρατήσεων και για την ορθή απεικόνιση ημερολογίου.

Ένας τρίτος άξονας είναι οι έλεγχοι που σχετίζονται με τη λήξη των holds, δεδομένου ότι το έργο έχει υλοποιήσει τη συνάρτηση expire_holds() και τη χρησιμοποιεί πριν από κρίσιμες σελίδες τόσο στη δημόσια πλευρά όσο και στο dashboard. Στα services, η expire_holds εντοπίζει Booking εγγραφές με status=PENDING και hold_expires_at στο παρελθόν και τις μετατρέπει σε REJECTED, καταγράφοντας αντίστοιχη ενέργεια. Η ύπαρξη αυτής της λειτουργίας απαιτεί δοκιμές που ελέγχουν την επίδραση στο get_events_for_range και στο availability_json. Ενα ΥΠΟ ΕΓΓΡΙΣΗ αίτημα με ληγμένο hold δεν πρέπει πλέον να εμφανίζεται ως PENDING_HOLD γεγονός και δεν πρέπει να μειώνει τη διαθεσιμότητα. Τέτοιες δοκιμές συνδέουν άμεσα τον πυρήνα (services) με το UI/UX, διότι αποτρέπουν ασυμφωνίες ανάμεσα σε αυτό που «πιστεύει» ο χρήστης και σε αυτό που αποθηκεύει η βάση στην πραγματικότηατα.

Ο τέταρτος άξονας είναι η ροή έγκρισης/απόρριψης από τον ιδιοκτήτη, η οποία στον κώδικα υλοποιείται με συνδυασμό dashboard views (dashboard/views.py:approve_booking_view, reject_booking_view) και υπηρεσιών (approve_booking, reject_booking). Η approve_booking εφαρμόζει transaction.atomic() και λαμβάνει κλείδωμα γραμμής μέσω select_for_update() όταν

ανακτά την κράτηση ωστέ να μειωθούν τα σενάρια όπου δύο ενέργειες (π.χ. δύο tabs ιδιοκτήτη) επιχειρούν να μεταβάλουν την ίδια κράτηση. Οι δοκιμές εδώ πρέπει να επιβεβαιώνουν ότι μόνο ΥΠΟ ΕΓΓΡΙΣΗ κρατήσεις μπορούν να εγκριθούν ή να απορριφθούν, ότι σε αποτυχία διαθεσιμότητας η έγκριση απορρίπτεται, και ότι σε επιτυχία παράγεται BookingLog εγγραφή με ActionType.APPROVE ή ActionType.REJECT αντίστοιχα. Παράλληλα, στο επίπεδο view, πρέπει να επιβεβαιώνεται ότι η ενέργεια απαιτεί POST και ότι η ανατροφοδότηση προς τον χρήστη γίνεται με messages.success ή messages.error, ώστε η διεπαφή να παραμένει προβλέψιμη.

Τέλος, ο πέμπτος άξονας δοκιμών αφορά τις «δευτερεύουσες» αλλά επιχειρησιακά σηματνικές ροές αλλαγών που είναι τα αιτήματα ακύρωσης και αιτήματα τροποποίησης. Στον κώδικα υπάρχουν οι υπηρεσίες request_cancellation, decide_cancellation_request, request_modification, decide_modification_request, καθώς και τα αντίστοιχα endpoints για τον πελάτη (reservations/views.py:request_cancellation_view, request_modification_view) και για τον ιδιοκτήτη (dashboard/views.py:decide_cancellation_view, decide_modification_view). Οι δοκιμές πρέπει να ελέγχουν ότι ο πελάτης μπορεί να δημιουργήσει αίτημα μόνο για κράτηση που του ανήκει (π.χ. η ανάκτηση της κράτησης γίνεται με φίλτρο customer=request.user), ότι το αίτημα εισέρχεται σε κατάσταση ChangeRequestStatus.PENDING, και ότι η απόφαση ιδιοκτήτη μεταβάλλει αντίστοιχα την κατάσταση αιτήματος και, όπου απαιτείται, την κατάσταση της κράτησης (π.χ. ακύρωση οδηγεί σε BookingStatus.CANCELLED). Ιδιαίτερα στην τροποποίηση, πρέπει να ελέγχεται ότι μια αποδεκτή τροποποίηση δεν δημιουργεί επικαλύψεις, δηλαδή ότι η υπηρεσία ελέγχει τη διαθεσιμότητα στο νέο διάστημα πριν εφαρμόσει αλλαγές. Στον κώδικα έχει προβλεφθεί αυτό με την χρήση του ignore_booking_id στην is_range_available ελέγχοντας την διαθεσιμότητα για μεταβολή της ίδιας κράτησης.

8.3 Έλεγχοι Ασφάλειας και Εξουσιοδότησης

Η ασφάλεια σε ένα σύστημα κρατήσεων σχετίζεται κυρίως με δύο άξονες:

- (α) την προστασία από ανεπιθύμητες ενέργειες και
- (β) την αποτροπή παραβίασης εξουσιοδότησης (authorization), όπου ένας χρήστης προσπαθεί να προσπελάσει ή να μεταβάλει δεδομένα που δεν του ανήκουν.

Στον κώδικα του έργου, οι κρίσιμες μεταβολές κατάστασης υλοποιούνται σε views που απαιτούν POST (π.χ. approve_booking_view, reject_booking_view, create_booking_request_view), πρακτική που συνεργάζεται με το Django CSRF middleware όπως προαναφέραμε ώστε να περιορίζεται ο κίνδυνος Cross-Site Request Forgery. Η ορθή χρήση CSRF tokens στα templates είναι αναγκαία προϋπόθεση, όπως τεκμηριώνεται στη σχετική οδηγία του Django, και η δοκιμή ασφάλειας εδώ μεταφράζεται σε έλεγχο ότι τα POST endpoints απορρίπτουν αιτήματα χωρίς έγκυρο CSRF token (σε περιβάλλον όπου το middleware είναι ενεργό) (Django Software Foundation, 2024, 2024e).

Ο δεύτερος και κρισιμότερος άξονας είναι η εξουσιοδότηση. Το έργο υλοποιεί authorization με δύο ισχυρά μοτίβα. Το πρώτο είναι οι decorators πρόσβασης στο dashboard, στο dashboard/views.py ο _owner_required επιβάλλει ότι μόνο ιδιοκτήτης/διαχειριστής μπορεί να εκτελέσει λειτουργίες έγκρισης/απόρριψης, απόφασης αιτημάτων αλλαγών και διαχείρισης blocks. Το δεύτερο είναι οι «ενσωματωμένοι» έλεγχοι εξουσιοδότησης σε queries του πελάτη, όπως φαίνεται στη reservations/views.py:booking_detail, όπου η ανάκτηση γίνεται με get_object_or_404(Booking, id=booking_id, customer=request.user). Αυτό το μοτίβο αντιμετωπίζει άμεσα το πρόβλημα Insecure Direct Object References (IDOR), ακόμη και αν ο χρήστης γνωρίζει ένα booking_id, δεν μπορεί να προσπελάσει κράτηση άλλου χρήστη, επειδή το φίλτρο customer αποτελεί τμήμα του query. Η δοκιμή ασφάλειας εδώ απαιτεί σενάρια όπου ένας χρήστης επιχειρεί να προσπελάσει ή να αιτηθεί ακύρωση/τροποποίηση σε κράτηση άλλου χρήστη, και το σύστημα πρέπει να απαντά με 404 ή με απόρριψη, χωρίς διαρροή πληροφορίας. Η σημασία τέτοιων δοκιμών επισημαίνεται και από το OWASP Web Security Testing Guide, το οποίο θεωρεί την εξουσιοδότηση και τους ελέγχους παράκαμψης (privilege escalation, IDOR) βασική κατηγορία ελέγχου (OWASP Foundation, 2023a, 2023b).

Τέλος, απαιτείται να ελεγχθεί ότι οι μεταβάσεις κατάστασης δεν μπορούν να προκληθούν έμμεσα από μη εξουσιοδοτημένη πλευρά. Για παράδειγμα, ένας πελάτης δεν πρέπει να μπορεί να καλέσει endpoints έγκρισης/απόρριψης που προορίζονται για ιδιοκτήτη, ενώ ένας ανώνυμος χρήστης δεν πρέπει να έχει πρόσβαση στις σελίδες “my bookings” ή να υποβάλει αιτήματα αλλαγών για κράτηση που δεν συνδέεται με λογαριασμό. Στο συγκεκριμένο έργο, η ύπαρξη ελέγχων και στο επίπεδο υπηρεσιών προσθέτει ένα δεύτερο στρώμα άμυνας, αποτρέποντας λογικά άκυρες ενέργειες ακόμη και αν ένα view κληθεί σε μη αναμενόμενο πλαίσιο.

ΚΕΦΑΛΑΙΟ 9ο

9. ΑΞΙΟΛΟΓΗΣΗ ΚΑΙ ΣΥΖΗΤΗΣΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ

9.1 Αντιστοίχιση Απαιτήσεων με Υλοποίηση

Η αξιολόγηση της παρούσας εφαρμογής πραγματοποιείται πρωτίστως μέσω της αντιστοίχισης των απαιτήσεων (Κεφάλαιο 2) με συγκεκριμένες υλοποιήσεις στον πηγαίο κώδικα. Στο έργο, οι βασικές λειτουργικές απαιτήσεις (παρουσίαση καταλυμάτων, ημερολόγιο διαθεσιμότητας, αίτημα κράτησης, ροή έγκρισης/απόρριψης) δεν αντιμετωπίζονται ως ανεξάρτητες “σελίδες”, αλλά ως συνεκτικές ροές που συνδέονται με κοινό επιχειρησιακό πυρήνα. Αυτό φαίνεται καθαρά από την επιλογή να συγκεντρωθούν κρίσιμες αποφάσεις στη μονάδα reservations/services.py και να χρησιμοποιούνται από πολλαπλά σημεία του συστήματος (δημόσια διεπαφή, API, dashboard). Για παράδειγμα, ο έλεγχος διαθεσιμότητας δεν είναι σε κάθε view, αλλά υλοποιείται κεντρικά στη συνάρτηση is_range_available, η οποία ορίζει τους κανόνες διαθεσιμότητας, μη διαθεσιμότητα όταν υπάρχει επικάλυψη με εγκεκριμένες κρατήσεις, με περιόδους μη διαθεσιμότητας (BlockedPeriod) και,υπό προϋποθέσεις,με ενεργές προσωρινές δεσμεύσεις εκκρεμών αιτημάτων. Στον κώδικα, η συνάρτηση τεκμηριώνει αυτή την πολιτική και εφαρμόζει ελέγχους επικάλυψης μέσω query φίλτρων.

Ένα ενδεικτικό απόσπασμα, που συνοψίζει τη λογική αυτή, είναι το παρακάτω:

```
def is_range_available(accommodation, check_in, check_out, *, ignore_booking_id=None) -> bool:
    if check_in >= check_out:
        return False

    approved_q = Q(status=BookingStatus.APPROVED)
    pending_hold_q = _effective_pending_filter()
```

```
    booking_qs = Booking.objects.filter(accommodation=accommodation).filter(approved_q | pending_hold_q)
    if ignore_booking_id is not None:
        booking_qs = booking_qs.exclude(id=ignore_booking_id)

    booking_qs = booking_qs.filter(check_in__lt=check_out, check_out__gt=check_in)
    if booking_qs.exists():
        return False

    block_qs = BlockedPeriod.objects.filter(accommodation=accommodation).filter(
        start_date__lt=check_out,
        end_date__gt=check_in,
    )
    if block_qs.exists():
        return False

    return True
```

Η παραπάνω υλοποίηση καλύπτει άμεσα την απαίτηση «κανόνες ακεραιότητας και επικάλυψης ημερομηνιών» (Κεφάλαιο 5), διότι αποτρέπει τη δημιουργία ή έγκριση κράτησης σε μη διαθέσιμο διάστημα. Ιδιαίτερη σημασία έχει η ύπαρξη του ορίσματος ignore_booking_id, το οποίο επιτρέπει επαναξιολόγηση διαθεσιμότητας χωρίς να "χτυπά" η ίδια η κράτηση πάνω στον εαυτό της. Το στοιχείο αυτό συνδέεται λειτουργικά με ροές που απαιτούν επανέλεγχο (π.χ. έγκριση ή πιθανή τροποποίηση κράτησης), και αποτελεί ένδειξη ότι η σχεδίαση δεν περιορίζεται σε ένα «ιδανικό σενάριο», αλλά λαμβάνει υπόψη πραγματικά σενάρια του κύκλου ζωής μιας κράτησης.

Η απαίτηση «υποβολή αιτήματος κράτησης» καλύπτεται μέσω της υπηρεσίας create_booking_request, η οποία δημιουργεί εγγραφή κράτησης σε κατάσταση PENDING και

εφαρμόζει μηχανισμό προσωρινής δέσμευσης μέσω hold_expires_at. Η ροή αυτή είναι βασική για την εμπειρία χρήστη και τη λειτουργική ορθότητα καθώς το σύστημα δεν μετατρέπει το αίτημα σε οριστική κράτηση πριν την απόφαση ιδιοκτήτη, αλλά παράλληλα αποτρέπει τη δημιουργία πολλών ανταγωνιστικών αιτημάτων στο ίδιο διάστημα, όταν η ρύθμιση PENDING_BLOCKS_AVAILABILITY είναι ενεργή. Ενδεικτικά, ο κώδικας εφαρμόζει την πολιτική ότι τα PENDING αιτήματα «μετρούν» ως δεσμευτικά μόνο όταν το hold τους είναι ενεργό, μέσω της _effective_pending_filter() που επιστρέφει ένα Q(status=PENDING, hold_expires_at__gt=now) όταν η σχετική ρύθμιση επιτρέπει το blocking των εκκρεμών αιτημάτων. Με αυτόν τον τρόπο, η εφαρμογή ευθυγραμμίζει τεχνικά την έννοια του «εκκρεμούς αιτήματος» με την έννοια της «προσωρινής δέσμευσης» που απαιτείται σε πραγματικά συστήματα κρατήσεων.

Παράλληλα, η απαίτηση «ροή έγκρισης/απόρριψης» καλύπτεται με σαφή και τυπικό τρόπο μέσω της approve_booking και της reject_booking, οι οποίες εκτελούνται εντός συναλλαγής (@transaction.atomic) και χρησιμοποιούν select_for_update() ώστε να μειώνεται ο κίνδυνος ασυνέπειας σε ανταγωνιστικές μεταβολές. Η έγκριση περιλαμβάνει επανέλεγχο διαθεσιμότητας, καθαρισμό ληγμένου hold (όταν χρειάζεται), μετάβαση κατάστασης σε APPROVED και ενημέρωση των πεδίων απόφασης (decided_at, decided_by, owner_notes). Αντίστοιχα, η απόρριψη μεταβαίνει σε REJECTED και μηδενίζει το hold. Σημαντικό στοιχείο αξιολόγησης είναι ότι κάθε κρίσιμη ενέργεια καταγράφεται σε BookingLog με συγκεκριμένο ActionType (π.χ. APPROVE, REJECT, EXPIRE_HOLD), ενισχύοντας την ιστορικότητα, στοιχείο που συνδέεται με μη λειτουργικές απαιτήσεις αξιοπιστίας και διαχειρισιμότητας.

9.2 Περιορισμοί και Παραδοχές Συστήματος

Παρότι η υλοποίηση καλύπτει συνεκτικά τις βασικές απαιτήσεις, η αξιολόγηση οφείλει να αναδείξει και τους εγγενείς περιορισμούς/παραδοχές του συστήματος, όπως προκύπτουν από τις σχεδιαστικές επιλογές του κώδικα. Ένας πρώτος ουσιώδης περιορισμός σχετίζεται με τη διαχείριση των εκκρεμών αιτημάτων (PENDING) και των holds. Στο expire_holds, η εφαρμογή δεν μετατρέπει ένα ληγμένο hold σε απόρριψη κράτησης, αλλά απλώς μηδενίζει το hold_expires_at και καταγράφει ενέργεια EXPIRE_HOLD. Αυτό σημαίνει ότι μπορεί να

υπάρχουν PENDING κρατήσεις στη βάση που δεν δεσμεύουν πλέον ημερομηνίες, αλλά παραμένουν ως εκκρεμείς εγγραφές.

Η επιλογή αυτή είναι απολύτως συνεπής με το σκεπτικό ότι η «εκκρεμότητα» αφορά την απόφαση ιδιοκτήτη και όχι κατ' ανάγκη μια χρονική δέσμευση, όμως δημιουργεί μια πρακτική παραδοχή ότι το σύστημα πρέπει να αντιμετωπίζει τέτοιες εγγραφές ως μη δεσμευτικές και να προσφέρει στους διαχειριστές τρόπο διαχείρισής τους (π.χ. απόρριψη, αρχειοθέτηση ή καθαρισμός). Ο κώδικας μερικώς καλύπτει το ζήτημα με το ότι οι ενεργές δεσμεύσεις εξαρτώνται από hold_expires_at > now, όμως η παρουσία πολλών "inactive pending" αιτημάτων μπορεί να επηρεάσει τη διαχειριστική εμπειρία (π.χ. λίστες εκκρεμών), αν δεν υπάρξει πολιτική φιλτραρίσματος ή αυτόματης απόφασης.

Δεύτερος περιορισμός αφορά την απόδοση και την κλιμάκωση. Ο έλεγχος διαθεσιμότητας βασίζεται σε queries που καταλήγουν σε exists() πάνω σε φίλτρα επικάλυψης για Booking και BlockedPeriod. Η προσέγγιση είναι ορθή για μικρό έως μεσαίο όγκο δεδομένων και αξιοποιεί την αποτελεσματικότητα του EXISTS σε SQL επίπεδο, ωστόσο η πραγματική απόδοση εξαρτάται από την ύπαρξη κατάλληλων δεικτών (indexes) στα πεδία accommodation, check_in, check_out, καθώς και στα start_date, end_date των blocks. Το Django, χωρίς ρητή δήλωση indexes, μπορεί να μην παράγει το βέλτιστο σχήμα για υψηλό φορτίο. Επομένως, μια παραδοχή του συστήματος είναι ότι λειτουργεί σε πλαίσιο όπου ο αριθμός καταλυμάτων και κρατήσεων δεν οδηγεί σε συστηματική συμφόρηση queries (Django Software Foundation, 2024).

Τρίτος περιορισμός αφορά τη διαχείριση ταυτόχρονης πρόσβασης (concurrency) και την "ισχυρή" αποτροπή διπλοκρατήσεων σε ακραία σενάρια. Η χρήση transaction.atomic και select_for_update() στην έγκριση/απόρριψη μειώνει σημαντικά τον κίνδυνο ασυνέπειας στη φάση απόφασης. Ωστόσο, η δημιουργία αιτήματος (create_booking_request) βασίζεται σε έλεγχο διαθεσιμότητας πριν την εισαγωγή στη βάση. Σε περιβάλλον υψηλού concurrency, δύο αιτήματα θα μπορούσαν θεωρητικά να περάσουν τον έλεγχο σχεδόν ταυτόχρονα και να δημιουργηθούν ως PENDING με ενεργό hold, εάν το blocking των pending holds είναι ενεργό και δεν υπάρχει πρόσθετος μηχανισμός μοναδικότητας ή "κρατήματος" σε επίπεδο βάσης. Το σύστημα μετριάζει το ζήτημα μέσω της πολιτικής holds και της επανεπιβεβαίωσης κατά την έγκριση, αλλά η παραδοχή είναι ότι η τελική συνέπεια εγγυάται στο στάδιο έγκρισης, όχι απαραίτητα στο στάδιο δημιουργίας αιτήματος, όταν η ταυτόχρονη δραστηριότητα είναι ακραία.

Για πλήρη αυστηρότητα σε περιβάλλον παραγωγής, θα μπορούσε να απαιτηθεί επιπλέον στρατηγική (π.χ. πιο επιθετικά locks ή ειδική μοντελοποίηση availability slots), κάτι που ξεπερνά το εύρος της τρέχουσας υλοποίησης (Bernstein, Hadzilacos & Goodman, 1987, Django Software Foundation, 2024).

Τέλος, ως περιορισμός πρέπει να συζητηθεί το επιχειρησιακό εύρος. Το σύστημα επικεντρώνεται στην ορθότητα κρατήσεων, στην παρουσίαση διαθεσιμότητας και στην έγκριση από ιδιοκτήτη. Δεν ενσωματώνει διαδικασίες πληρωμής, δυναμική τιμολόγηση ανά περίοδο, κανόνες ακυρωτικών πολιτικών (πέρα από το αίτημα ακύρωσης) ή μηχανισμούς ειδοποίησης (email/SMS). Η απουσία αυτών δεν αποτελεί τεχνική αδυναμία, αλλά σχεδιαστική παραδοχή που συνδέεται με τον στόχο της εργασίας. Παράλληλα, ο κώδικας ήδη θέτει κατάλληλες βάσεις επέκτασης, διότι ο κύκλος ζωής κρατήσεων μοντελοποιείται μέσω statuses και logs, και οι ροές αλλαγών (cancellation/modification) παρέχουν ένα πλαίσιο για μελλοντική ενσωμάτωση πολιτικών και αυτοματισμών.

ΚΕΦΑΛΑΙΟ 10ο

10. ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ

10.1 Συμπεράσματα

Η παρούσα εργασία κατέληξε στην ανάπτυξη ενός ολοκληρωμένου και λειτουργικού συστήματος κρατήσεων για τουριστικά καταλύματα, το οποίο καλύπτει με συνέπεια τον βασικό κύκλο ζωής μιας κράτησης, από την παρουσίαση των καταλυμάτων και την προβολή διαθεσιμότητας, έως την υποβολή αιτήματος, τη διαχειριστική απόφαση και τη διαχείριση ακυρώσεων ή τροποποιήσεων. Από την ανάλυση που προηγήθηκε προκύπτει ότι το σύστημα δεν περιορίζεται σε μια απλή παρουσίαση πληροφοριών, αλλά ενσωματώνει σαφείς και συνεπείς κανόνες λειτουργίας, διασφαλίζοντας την ορθότητα των αποτελεσμάτων που παρέχει στους χρήστες.

Κεντρικό συμπέρασμα της εργασίας αποτελεί το γεγονός ότι η διαθεσιμότητα αντιμετωπίζεται ως δυναμικό αποτέλεσμα επιχειρησιακών κανόνων και όχι ως στατική πληροφορία. Η προσέγγιση αυτή επιτρέπει στο σύστημα να ανταποκρίνεται ορθά σε μεταβαλλόμενες συνθήκες, όπως νέες κρατήσεις, διαχειριστικές αποφάσεις ή χρονικούς αποκλεισμούς, και να αποτρέπει ασυνέπειες που θα μπορούσαν να οδηγήσουν σε λανθασμένες ή αντικρουόμενες κρατήσεις.

Επιπλέον, η σαφής διαχείριση καταστάσεων και ρόλων συμβάλλει ουσιαστικά στην αξιοπιστία της εφαρμογής. Οι διαφορετικοί τύποι χρηστών διαθέτουν ξεκάθαρα καθορισμένες δυνατότητες και περιορισμούς, γεγονός που ενισχύει τόσο την ορθή λειτουργία του συστήματος όσο και την εμπιστοσύνη των εμπλεκομένων μερών. Η ύπαρξη ελεγχόμενων ροών λήψης αποφάσεων και καταγραφής ενεργειών προσδίδει στο σύστημα διαφάνεια και δυνατότητα ελέγχου, χαρακτηριστικά ιδιαίτερα σημαντικά σε πραγματικά επιχειρησιακά περιβάλλοντα.

Τέλος, από τη συνολική αξιολόγηση προκύπτει ότι, παρότι το σύστημα καλύπτει με επιτυχία τους βασικούς στόχους της εργασίας, αναδεικνύονται και σημεία που θα μπορούσαν να βελτιωθούν σε ένα πιο ώριμο περιβάλλον παραγωγής. Ορισμένες συμπεριφορές αποτελούν συνειδητές σχεδιαστικές επιλογές που εξυπηρετούν τη λειτουργικότητα και την ευελιξία, ωστόσο θα μπορούσαν να ενισχυθούν με πρόσθετες πολιτικές διαχείρισης.

10.2 Προτάσεις Μελλοντικής Εξέλιξης

Οι μελλοντικές επεκτάσεις του συστήματος μπορούν να οργανωθούν γύρω από τρεις άξονες:

1. επέκταση επιχειρησιακού πεδίου
2. ενίσχυση τεχνικής ωριμότητας
3. αναβάθμιση εμπειρίας χρήστη

Ως προς το επιχειρησιακό πεδίο, μια πρώτη σημαντική επέκταση αφορά τη διαχείριση πληρωμών και πολιτικών ακύρωσης. Στην τρέχουσα υλοποίηση, ο κύκλος ζωής της κράτησης ολοκληρώνεται στη φάση έγκρισης/απόρριψης και σε αιτήματα ακύρωσης/τροποποίησης που αποφασίζονται από τον ιδιοκτήτη. Ωστόσο, η ενσωμάτωση πληρωμών (π.χ. προκαταβολή ή πλήρης εξόφληση) θα απαιτούσε νέες οντότητες και καταστάσεις, καθώς και μηχανισμούς ασφαλούς καταγραφής συναλλαγών. Η αρχιτεκτονική του έργου είναι ήδη ευνοϊκή για τέτοια επέκταση, διότι οι μεταβάσεις κατάστασης είναι κεντρικοποιημένες σε services.

Έτσι, η προσθήκη καταστάσεων όπως “awaiting payment” ή “paid” θα μπορούσε να ενσωματωθεί με ελεγχόμενο τρόπο, ενώ το BookingLog μπορεί να επεκταθεί με νέα ActionType για γεγονότα πληρωμής.

Ως προς την τεχνική ωριμότητα, Οοέλεγχος διαθεσιμότητας βασίζεται σε ερωτήματα επικάλυψης πάνω σε Booking και BlockedPeriod. Σε μικρό αριθμό εγγραφών, αυτό είναι αποτελεσματικό, ιδίως όταν χρησιμοποιείται exists() που τερματίζει νωρίς. Ωστόσο, για περιβάλλον με μεγαλύτερο όγκο κρατήσεων, προτείνεται η εισαγωγή ρητών δεικτών (indexes) στα πεδία που χρησιμοποιούνται συστηματικά σε φίλτρα (accommodation, check_in, check_out, start_date, end_date, καθώς και status). Η Django μοντελοποίηση επιτρέπει την προσθήκη

indexes = [...] στα Meta των models, κάτι που θα βελτίωνε τον χρόνο εκτέλεσης των queries χωρίς να μεταβληθεί η επιχειρησιακή λογική. Παράλληλα, θα μπορούσε να εξεταστεί caching επιπέδου "events for month" για την αρχική σελίδα, δεδομένου ότι στο core/views.py ο υπολογισμός διαθεσιμότητας για κάθε κατάλυμα και ημέρα μπορεί να αυξηθεί υπολογιστικά σε μεγάλες κλίμακες. Η εισαγωγή cache, με προσεκτικό invalidation όταν αλλάζουν κρατήσεις/blocks, αποτελεί μια χρήσιμη τεχνική εξέλιξη.

Ως προς την εμπειρία χρήστη, προτείνεται η περαιτέρω βελτίωση της διαχειριστικής διεπαφής (dashboard) ώστε να αντιμετωπίζεται το ζήτημα των "inactive pending" αιτημάτων. Η ίδια η υλοποίηση προσφέρει τα δεδομένα για κάτι τέτοιο, το Booking διαθέτει hold_expires_at, και η μέθοδος is_hold_active() μπορεί να χρησιμοποιηθεί ώστε το UI να διαφοροποιεί οπτικά τα αιτήματα που δεσμεύουν ημερομηνίες από εκείνα που απλώς αναμένουν απόφαση χωρίς ενεργό hold. Επιπλέον, το dashboard θα μπορούσε να παρέχει φίλτρα για "pending with active hold" και "pending without hold", ή να προτείνει αυτοματοποιημένη απόφαση (π.χ. auto-reject) μετά από συγκεκριμένο χρονικό διάστημα, επιλογή που μπορεί να υλοποιηθεί με scheduled job χωρίς να διαταράξει τον πυρήνα της λογικής.

Συνολικά, οι προτάσεις μελλοντικής εξέλιξης δεν απαιτούν αναδόμηση του συστήματος, αλλά μπορούν να ενσωματωθούν σταδιακά λόγω της υπάρχουσας δομής. Η κεντρικοποίηση των κανόνων σε υπηρεσίες, η χρήση statuses και logs, και ο σαφής διαχωρισμός ρόλων δημιουργούν ένα στιβαρό πλαίσιο επέκτασης. Έτσι, το αποτέλεσμα καταλήγει όχι μόνο σε μια ολοκληρωμένη εφαρμογή πρωτοτύπου, αλλά και σε μια τεχνικά τεκμηριωμένη βάση που μπορεί να εξελιχθεί προς μια πιο πλήρη μηχανή κρατήσεων, με αυξημένες απαιτήσεις ασφάλειας, απόδοσης και επιχειρησιακού βάθους.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Airbnb. (χ.χ.). How to book a home: Instant Book and reservation requests. https://www.airbnb.com/help/article/85

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. Communications of the ACM, 26(11), 832–843. https://doi.org/10.1145/182.358434

Bernstein, P. A., Hadzilacos, V., & Goodman, N. (1987). Concurrency control and recovery in database systems. Addison-Wesley.

Booking.com. (χ.χ.). Understanding our commission. https://partner.booking.com/en-us/help/commission-invoices-tax/commission/understanding-our-commission

Django Software Foundation. (2024). *Django documentation*. https://docs.djangoproject.com/

Evans, E. (2003). *Domain-driven design: Tackling complexity in the heart of software*. Addison-Wesley.

Fowler, M. (2002). *Patterns of enterprise application architecture*. Addison-Wesley.

HotelDruid. (χ.χ.). HotelDruid: Open source property management system. https://www.hoteldruid.com/en/

International Organization for Standardization. (2011). *ISO/IEC 25010: Systems and software engineering—Systems and software Quality Requirements and Evaluation (SQuaRE)*. https://www.iso.org/standard/35733.html

Nielsen, J. (2024). 10 usability heuristics for user interface design (αρχική δημοσίευση 1994). Nielsen Norman Group. https://www.nngroup.com/articles/ten-usability-heuristics/

Object Management Group. (2017). OMG Unified Modeling Language (OMG UML), version 2.5.1. https://www.omg.org/spec/UML/2.5.1/

OWASP Foundation. (2023a). *OWASP Top 10—Web application security risks*. https://owasp.org/www-project-top-ten/

OWASP Foundation. (2023b). *OWASP web security testing guide*. https://owasp.org/www-project-web-security-testing-guide/

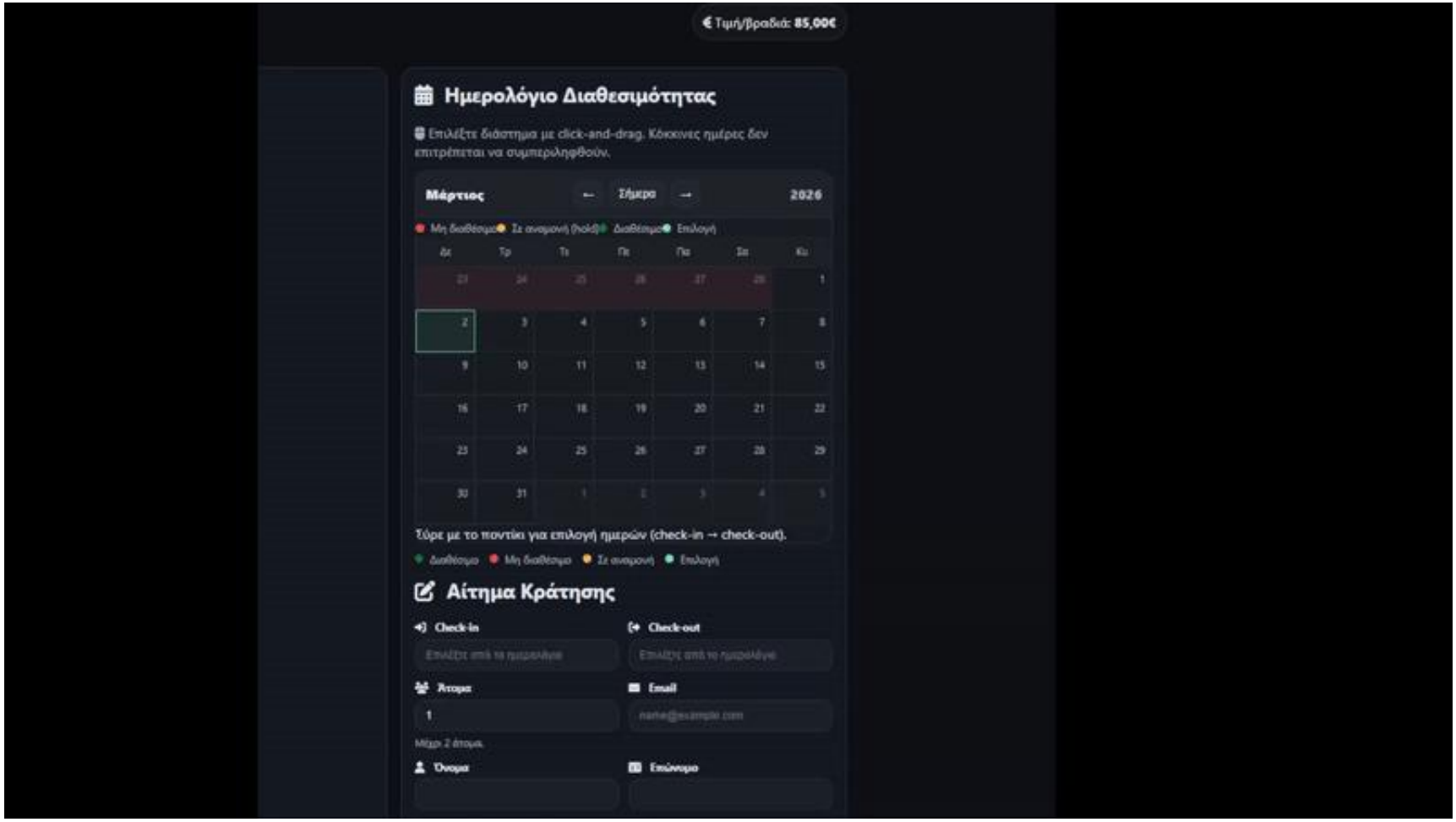
QloApps. (χ.χ.). QloApps: Free and open source hotel management system. https://www.qloapps.com/

Smoobu. (χ.χ.). Prices. https://www.smoobu.com/en/prices/

Sommerville, I. (2016). Software engineering (10th ed.). Pearson.

ΠΑΡΑΡΤΗΜΑ Α

Στιγμιότυπο παρουσίασης UI του συστήματος



Εικόνα 15 Επιλογή ημερομηνιών για κράτηση