ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόγραμμα Μεταπτυχιακών Σπουδών

«ΠΡΟΗΓΜΕΝΑ ΣΥΣΤΗΜΑΤΑ ΠΛΗΡΟΦΟΡΙΚΗΣ - ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ ΚΑΙ ΤΕΧΝΗΤΗΣ ΝΟΗΜΟΣΥΝΗΣ»

Μεταπτυχιακή διατριβή

Τίτλος Μεταπτυχιακής Διατριβής	Σύστημα Διαχείρισης Αιτημάτων Υποστήριξης με Έλεγχο Πρόσβασης βάσει Ρόλων A Support Ticket Management System with Role-Based Access Control
Ονοματεπώνυμο Φοιτητή	Αθανάσιος Ευσταθίου
Πατρώνυμο	Γεώργιος
Αριθμός Μητρώου	ΜΠΣΠ2310
Επιβλέπων	Ευάγγελος Σακκόπουλος, Αναπληρωτής Καθηγητής

Ημερομηνία Παράδοσης Ιανουάριος 2026

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Ευάγγελος Σακκόπουλος, Αναπληρωτής Καθηγητής	Διονύσιος Σωτηρόπουλος Αναπληρωτής Καθηγητής	Ιωάννης Τασούλας, Επίκουρος Καθηγητής

Copyright

Ευχαριστίες

Με την ολοκλήρωση της παρούσας μεταπτυχιακής διατριβής, θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες σε όσους συνέβαλαν, με τον δικό τους τρόπο, στην επιτυχή της περάτωση.

Πρωτίστως, οφείλω να ευχαριστήσω τον επιβλέποντα καθηγητή μου, Ευάγγελο Σακκόπουλο, για την ανάθεση του θέματος, την εμπιστοσύνη που μου έδειξε και την καθοδήγησή του σε όλη τη διάρκεια της εκπόνησης της εργασίας. Η ακαδημαϊκή του εποπτεία ήταν πολύτιμη για την τήρηση των προδιαγραφών και την ολοκλήρωση αυτού του έργου.

Η μεγαλύτερη ευγνωμοσύνη μου, ωστόσο, ανήκει στην οικογένειά μου. Η αμέριστη υποστήριξη, η υπομονή και η συνεχής ενθάρρυνση που μου παρείχαν, ειδικά στις πιο απαιτητικές περιόδους, αποτέλεσαν το θεμέλιο πάνω στο οποίο στηρίχθηκε όλη αυτή η προσπάθεια. Χωρίς την αγάπη και την πίστη τους, το ταξίδι αυτό θα ήταν ασύγκριτα δυσκολότερο.

Τέλος, ένα μεγάλο ευχαριστώ στους φίλους μου, που με την κατανόηση και το χιούμορ τους μου προσέφεραν μια αναγκαία απόδραση από την πίεση της συγγραφής και με βοήθησαν να διατηρήσω την ισορροπία μου.

Περίληψη

Η παρούσα μεταπτυχιακή διατριβή παρουσιάζει τη σχεδίαση και υλοποίηση ενός ολοκληρωμένου συστήματος διαχείρισης αιτημάτων υποστήριξης. Βασικός στόχος ήταν η ανάπτυξη μιας full-stack web εφαρμογής που παρέχει ένα κεντρικοποιημένο περιβάλλον για την παρακολούθηση και επίλυση αιτημάτων, με την ονομασία της εφαρμογής και το λογότυπο να είναι δυναμικά παραμετροποιήσιμα από τον διαχειριστή.

Η αρχιτεκτονική του συστήματος βασίζεται σε ένα ασφαλές RESTful API που υλοποιήθηκε με Spring Boot, και σε ένα responsive frontend τύπου Single-Page Application (SPA) που αναπτύχθηκε με React. Η ασφάλεια επιτυγχάνεται μέσω JSON Web Tokens (JWT) και ενός εξελιγμένου μοντέλου Ελέγχου Πρόσβασης βάσει Ρόλων (RBAC) που υποστηρίζει τέσσερις διακριτούς ρόλους. Για την ακεραιότητα των δεδομένων, χρησιμοποιήθηκε μια σχεσιακή βάση δεδομένων MySQL με Spring Data JPA. Ένα από τα βασικά τεχνικά χαρακτηριστικά του συστήματος είναι η υλοποίηση ενός event-driven μηχανισμού καταγραφής ενεργειών (auditing) με χρήση @TransactionalEventListener, ο οποίος διασφαλίζει την ακεραιότητα του ιστορικού των αιτημάτων χωρίς να προκαλεί προβλήματα ταυτοχρονισμού.

Επιστημονική Περιοχή: *Μηχανική Λογισμικού, Ανάπτυξη Διαδικτυακών Εφαρμοών, Ασφάλεια Πληροφοριακών Συστημάτων.*

Λέξεις-Κλειδιά: *Spring Boot, React, REST API, JSON Web Token (JWT), Έλεγχος Πρόσβασης βάσει Ρόλων (RBAC).*

Abstract

This master's thesis presents the design and implementation of a full-stack support ticket management system. A primary objective was the development of a web application that provides a centralized environment for tracking and resolving requests, featuring a dynamically configurable application name and logo manageable by an administrator.

The system's architecture is based on a secure RESTful API implemented with Spring Boot and a responsive Single-Page Application (SPA) frontend developed with React. Security is achieved through JSON Web Tokens (JWT) and an advanced Role-Based Access Control (RBAC) model supporting four distinct user roles. For data persistence, a relational MySQL database is utilized with Spring Data JPA. A key technical feature is the implementation of an event-driven auditing mechanism using @TransactionalEventListener, which ensures the integrity of the ticket history without introducing concurrency issues.

Scientific Area: *Software Engineering, Web Application Development, Information Systems Security.*

Keywords: *Spring Boot, React, REST API, JSON Web Token (JWT), Role-Based Access Control (RBAC).*

Πίνακας Περιεχομένων

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Κατάλογος Διαγραμμάτων

1. Εισαγωγή

1.1. Περιγραφή και σκοπός

Η αποτελεσματική διαχείριση των λειτουργιών υποστήριξης πελατών αποτελεί μια κρίσιμη λειτουργία για τις σύγχρονες επιχειρήσεις λογισμικού. Σε μια ανταγωνιστική αγορά, η ποιότητα της εξυπηρέτησης πελατών είναι ένας βασικός παράγοντας διαφοροποίησης, που επηρεάζει άμεσα τη διατήρηση των πελατών και την επιχειρηματική επιτυχία, μια αρχή κεντρικής σημασίας για τη **Διαχείρισης Σχέσεων Πελατών (Customer Relationship Management - CRM)** [1]. Καθώς οι οργανισμοί μεγαλώνουν, η εξάρτηση από ad-hoc εργαλεία όπως τα κοινόχρηστα εισερχόμενα email για τη διαχείριση αιτημάτων υποστήριξης γίνεται όλο και πιο αβάσιμη. Αυτή η αδόμητη προσέγγιση στερείται της υπευθυνότητας και των δεδομένων που είναι απαραίτητα για την αποτελεσματική διαχείριση της ροής εργασίας. Η ανάπτυξη ενός ειδικού συστήματος υποστήριξης είναι επομένως μια σημαντική πρόκληση για τη μηχανική λογισμικού, που απαιτεί μια συστηματική και πειθαρχημένη προσέγγιση για να γεφυρωθεί το χάσμα μεταξύ των αναγκών των πελατών και των επιχειρηματικών διαδικασιών [2]. Η διεθνής βιβλιογραφία επισημαίνει βασικές κατευθύνσεις έρευνας σε αυτόν τον τομέα, συμπεριλαμβανομένου του σχεδιασμού επεκτάσιμων αρχιτεκτονικών συστημάτων και της εφαρμογής ισχυρών μοντέλων ασφάλειας, όπως ο **έλεγχος πρόσβασης βάσει ρόλων (RBAC)**, για τη διασφάλιση της ακεραιότητας των δεδομένων [3].

Ο κεντρικός στόχος αυτής της διατριβής είναι ο σχεδιασμός και η εφαρμογή μιας ολοκληρωμένης, πλήρους εφαρμογής ιστού που θα λειτουργεί ως κεντρική πλατφόρμα για τη διαχείριση ολόκληρου του κύκλου ζωής των αιτημάτων υποστήριξης. Η μεθοδολογική προσέγγιση βασίζεται στις αρχές μιας σύγχρονης **Τριεπίπεδης Αρχιτεκτονικής**, η οποία διαχωρίζει καθαρά τα θέματα της εφαρμογής σε ξεχωριστά επίπεδα [4].Αυτό επιτυγχάνεται μέσω της ανάπτυξης ενός ασφαλούς, stateless **RESTful API** για το backend, ενός αρχιτεκτονικού στυλ που ορίζεται από τους περιορισμούς του statelessness και μιας ομοιόμορφης διεπαφής [5].Για το frontend, επιλέχθηκε μια αρχιτεκτονική **Single-Page Application (SPA)**. Αυτό το σύγχρονο πρότυπο ανάπτυξης ιστού παρέχει μια πιο ρευστή και ανταποκρινόμενη εμπειρία χρήστη, αποδίδοντας δυναμικά τις προβολές στον browser, αντιπροσωπεύοντας μια σημαντική εξέλιξη από τις παραδοσιακές εφαρμογές που αποδίδονται από τον server [6].

Για την επίτευξη αυτού του πρωταρχικού στόχου, καθορίστηκαν συγκεκριμένοι στόχοι. Ένας βασικός στόχος ήταν η εφαρμογή ενός εξελιγμένου μοντέλου RBAC για την υποστήριξη τεσσάρων διακριτών ρόλων χρηστών, ο καθένας με μια προσαρμοσμένη εμπειρία χρήστη και ένα λεπτομερές σύνολο δικαιωμάτων. Περαιτέρω στόχοι περιλάμβαναν την ανάπτυξη ενός δυναμικού συστήματος φιλτραρίσματος και σελιδοποίησης από την πλευρά του διακομιστή, την ενσωμάτωση ενός ασφαλούς μηχανισμού επισύναψης αρχείων και τη δημιουργία ενός πίνακα ελέγχου με αναγνώριση ρόλων. Μια σημαντική τεχνική συμβολή αυτής της εργασίας είναι η υλοποίηση ενός ισχυρού, βασισμένου σε συμβάντα, συστήματος ελέγχου που δημιουργεί ένα αμετάβλητο ιστορικό όλων των αλλαγών κατάστασης των εισιτηρίων, σχεδιασμένο να είναι ασφαλές από συναλλακτική άποψη και αποσυνδεδεμένο από την βασική επιχειρηματική λογική. Η επιτυχής υλοποίηση αυτών των στόχων παρέχει ένα λειτουργικό πρωτότυπο επιχειρηματικού επιπέδου που χρησιμεύει ως πρακτική επίδειξη της εφαρμογής σύγχρονων αρχών μηχανικής λογισμικού.

Η παρούσα διατριβή έχει την ακόλουθη δομή: **Το κεφάλαιο 2** περιγράφει λεπτομερώς την ανάλυση και το σχεδιασμό του συστήματος, συμπεριλαμβανομένων των απαιτήσεων, των ρόλων των χρηστών και των τεχνολογικών αιτιολογήσεων. Το **Κεφάλαιο 3** παρουσιάζει τη λεπτομερή αρχιτεκτονική του συστήματος, καλύπτοντας το σχήμα της βάσης δεδομένων, το σχεδιασμό του REST API και το μοντέλο ασφάλειας. Τα **Κεφάλαια 4 και 5** τεκμηριώνουν την τεχνική υλοποίηση του backend και του frontend, αντίστοιχα. Το **Κεφάλαιο 6** επιδεικνύει τη λειτουργικότητα της εφαρμογής μέσω διαφόρων ροών εργασίας των χρηστών. Τέλος, το **Κεφάλαιο 7** συνοψίζει την εργασία, συζητά τους περιορισμούς της και προτείνει κατευθύνσεις για μελλοντικές βελτιώσεις.

1.2. Παραδείγματα

Οι προκλήσεις της μη δομημένης διαχείρισης της υποστήριξης δεν είναι θεωρητικές, αλλά εκδηλώνονται σε καθημερινά προβλήματα σε ένα ευρύ φάσμα επιχειρήσεων που βασίζονται σε λογισμικό. Το σύστημα έχει σχεδιαστεί για να παρέχει μια ευέλικτη λύση σε αυτές τις προκλήσεις. Για να γίνει κατανοητή η πρακτική σημασία και η ευρεία εφαρμογή του, εξετάζονται τα παρακάτω παραδείγματα χρήσης της εφαρμογής:

1. **Μια εφαρμογή για κινητά B2C (Business-to-Consumer)**

- **Σενάριο:** Μια εταιρεία αναπτύσσει ένα δημοφιλές παιχνίδι για κινητά ή μια εφαρμογή παραγωγικότητας με χιλιάδες μεμονωμένους χρήστες. Η ομάδα υποστήριξης είναι μικρή και κατακλύζεται από μεγάλο όγκο διαφορετικών αιτημάτων.
- **Πρόβλημα χωρίς το σύστημα:** Το ηλεκτρονικό ταχυδρομείο υποστήριξης γεμίζει από αναφορές σφαλμάτων, προτάσεις για νέες λειτουργίες και ερωτήσεις σχετικά με τη χρέωση. Ένα επείγον ζήτημα σχετικά με έναν χρήστη που δεν μπορεί να πραγματοποιήσει μια αγορά εντός της εφαρμογής χάνεται ανάμεσα σε εκατοντάδες λιγότερο κρίσιμα email. Δεν υπάρχει τρόπος να δοθεί προτεραιότητα στα ζητήματα, να παρακολουθούνται τα επαναλαμβανόμενα σφάλματα ή να διασφαλιστεί ότι κάθε χρήστης θα λάβει έγκαιρη απάντηση.
- **Λύση με την εφαρμογή:**
    - Ένας πελάτης υποβάλλει ένα ticket απευθείας μέσω της εφαρμογής ή ενός διαδικτυακού portal, κατηγοριοποιώντας το ως ζήτημα «Χρέωσης και Τιμολόγησης» με «Υψηλή» προτεραιότητα.
    - Ένας υπεύθυνος βλέπει το ticket υψηλής προτεραιότητας στην ουρά «Μη εκχωρημένα» του πίνακα ελέγχου του και το εκχωρεί αμέσως σε έναν υπάλληλο υποστήριξης.
    - Ο υπάλληλος μπορεί να δει το ticket, να επικοινωνήσει με τον πελάτη και να επιλύσει το πρόβλημα πληρωμής, με ολόκληρη τη συνομιλία και όλες τις αλλαγές κατάστασης να καταγράφονται σε ένα αμετάβλητο αρχείο καταγραφής ελέγχου.

2. **Μια B2B (Business-to-Business) SaaS πλατφόρμα**

- **Σενάριο:** Μια εταιρεία παρέχει μια πλατφόρμα Software-as-a-Service (SaaS) σε άλλες επιχειρήσεις. Κάθε εταιρεία-πελάτης έχει πολλούς υπαλλήλους που ενδέχεται να χρειαστούν υποστήριξη. Η ομάδα υποστήριξης είναι οργανωμένη και πρέπει να διαχειρίζεται σύνθετα τεχνικά ζητήματα.
- **Πρόβλημα χωρίς το σύστημα:** Ένα κρίσιμο σφάλμα που επηρεάζει έναν σημαντικό πελάτη αναφέρθηκε από έναν από τους νεότερους υπαλλήλους του μέσω email σε έναν

συγκεκριμένο πράκτορα που τυχαία βρισκόταν σε διακοπές. Το πρόβλημα παραμένει απαρατήρητο για μέρες, παραβιάζοντας τη Συμφωνία Επιπέδου Υπηρεσιών (SLA) και θέτοντας σε κίνδυνο τη σχέση με τον πελάτη. Τα μέλη της εσωτερικής ομάδας δεν μπορούν να δουν την κατάσταση του προβλήματος.

- **Λύση με την πλατφόρμα:**
    - Ένας υπάλληλος της εταιρείας-πελάτη υποβάλλει ένα ticket, κατηγοριοποιώντας το ως «Τεχνική Υποστήριξη» και επισυνάπτοντας αρχεία καταγραφής του διακομιστή.
    - Το ticket είναι ορατό σε ολόκληρη την ομάδα υποστήριξης. Ένας διαθέσιμος υπάλληλος υποστήριξης διεκδικεί αμέσως το ticket, αλλάζοντας την κατάστασή του σε «Ανοιχτό».
    - Ο πράκτορας διερευνά το ζήτημα, αλλά χρειάζεται βοήθεια. Προσθέτει μια **εσωτερική σημείωση**, επισημαίνοντας έναν ανώτερο προγραμματιστή: «@senior_dev, μπορείς να δεις τα συνημμένα αρχεία καταγραφής; Φαίνεται να είναι ένα ζήτημα βάσης δεδομένων». Αυτή η επικοινωνία δεν είναι ορατή στον πελάτη.
    - Ένας ADMIN μπορεί αργότερα να δημιουργήσει μια αναφορά για όλα τα εισιτήρια από αυτόν τον συγκεκριμένο πελάτη ή κατηγορία για να αναλύσει επαναλαμβανόμενα προβλήματα.

3. **Εσωτερικό τμήμα υποστήριξης IT**

- **Σενάριο:** Μια μεγάλη εταιρεία χρησιμοποιεί την εφαρμογή εσωτερικά για να διαχειρίζεται αιτήματα υποστήριξης IT για τους δικούς της υπαλλήλους.
- **Πρόβλημα χωρίς την εφαρμογή:** Ο φορητός υπολογιστής ενός υπαλλήλου χαλάει. Στέλνει ένα email στο τμήμα IT. Το αίτημα δεν καταγράφεται επίσημα, δεν υπάρχει αριθμός αιτήματος για αναφορά και ο υπάλληλος δεν έχει καμία εικόνα για το αν το αίτημά του βρίσκεται σε επεξεργασία ή ποιος είναι υπεύθυνος για αυτό.
- **Λύση με την εφαρμογή:**
    - Ένας υπάλληλος (πελάτης) δημιουργεί ένα αίτημα με θέμα «Ο φορητός υπολογιστής δεν ανάβει».
    - Ο υπεύθυνος IT βλέπει το αίτημα και το αναθέτει σε έναν συγκεκριμένο υπάλληλο υποστήριξης IT που ειδικεύεται σε ζητήματα υλικού.
    - Ο υπεύθυνος ενημερώνει την κατάσταση του εισιτηρίου σε «Σε εξέλιξη» και ο υπάλληλος μπορεί να δει αυτήν την ενημέρωση στον προσωπικό του πίνακα ελέγχου, με αποτέλεσμα να νιώθει σιγουριά ότι το πρόβλημά του αντιμετωπίζεται.

Αυτά τα παραδείγματα δείχνουν πώς ένα δομημένο, βασισμένο σε ρόλους σύστημα διαχείρισης εισιτηρίων όπως αυτό που αναφέρει η διατριβή, παρέχει τα απαραίτητα εργαλεία για λογοδοσία, ορατότητα και αποτελεσματική διαχείριση ροής εργασίας σε μια ποικιλία πραγματικών επιχειρηματικών συνθηκών.

2. Ανάλυση και Σχεδίαση Συστήματος

2.1. Ανάλυση Απαιτήσεων

Η αρχική φάση του σχεδιασμού του συστήματος ήταν μια ολοκληρωμένη ανάλυση των λειτουργικών και μη λειτουργικών απαιτήσεων για μια σύγχρονη πλατφόρμα διαχείρισης εισιτηρίων υποστήριξης. Αυτή η διαδικασία περιλάμβανε τον προσδιορισμό των βασικών προβλημάτων που έπρεπε να επιλυθούν και τον καθορισμό των συγκεκριμένων δυνατοτήτων που έπρεπε να διαθέτει το σύστημα για να τα αντιμετωπίσει.

Για να καθοριστεί μια βάση αναφοράς των βασικών χαρακτηριστικών και των βέλτιστων πρακτικών, πραγματοποιήθηκε μια **συγκριτική ανάλυση** των υφιστάμενων, κορυφαίων πλατφορμών του κλάδου. Για την ανάλυση αυτή επιλέχθηκαν τρία σημαντικά συστήματα: το **Jira Service Management** [7], το **Zendesk** [8], το **Microsoft Dynamics 365 Customer Service** [9] και το HubSpot [10]. Οι πλατφόρμες αυτές αξιολογήθηκαν προκειμένου να προσδιοριστούν οι κοινές αρχιτεκτονικές δομές και οι βασικές λειτουργίες που θεωρούνται πρότυπα του κλάδου. Τα αποτελέσματα της ανάλυσης αυτής συνοψίζονται στον Πίνακας 2.1.

Πίνακας 2.1 Συγκριτική ανάλυση των βασικών χαρακτηριστικών επιχειρηματικών εφαμοργών υποστήριξης

Λειτουργία / Δυνατότητα	Jira Service Management	Zendesk	Dynamics 365	Hubspot
Βασική λειτουργία αιτημάτων				
Δημιουργία και κύκλος ζωής εισιτηρίων	☑ Requests / issues	☑ Tickets	☑ Κύκλος ζωής μέσω workflows	☑ Service Hub tickets
Συζητήσεις σχολίων	☑ Σχόλια / εσωτερικές σημειώσεις	☑Σχόλια (δημόσια/ιδιωτικά)	☑ Σημειώσεις & timeline	☑ Σχόλια σε εισιτήρια
Συνημμένα αρχεία	☑	☑ Συνημμένα σε tickets	☑ Συνημμένα σε cases	☑ Ανέβασμα αρχείων
Οργάνωση και ροή εργασίας				
Έλεγχος πρόσβασης βάσει ρόλου (RBAC)	☑ Ρόλοι + Atlassian groups	☑ Ρόλοι (agent/admin/user)	☑ Enterprise RBAC (Azure AD)	☑ Ρόλοι & δικαιώματα
Ανάθεση εισιτηρίων (χειροκίνητη/αυτόματη)	☑ Χειροκίνητη + αυτοματισμοί	☑ Χειροκίνητη + triggers/macros	☑ Κανόνες ανάθεσης	☑ Χειροκίνητη & αυτόματη
Δυναμικές/προσαρμόσιμες κατηγορίες	☑ Custom fields, request types	☑ Custom fields, tags	☑ Πλήρης παραμετροποίηση οντοτήτων	☑ Custom properties/tags
Έλεγχος και ιστορικό				
Αμετάβλητο αρχείο ιστορικού	☑ Αμετάβλητο	☑ Audit logs (όχι πλήρως αμετάβλητα)	☑ Αμετάβλητα logs (compliance)	☑ Logs αλλά επεξεργάσιμες σημειώσεις

	ιστορικό issue			
Αποσυνδεδεμένος έλεγχος βάσει συμβάντων	☒ Webhooks & audit events	☒ Webhooks, μερικώς	☑ Event-driven logging	☑ Webhooks, περιορισμένο
RESTful API ως κύρια διεπαφή	☑ Ναι	☑ Ναι	☑ Ναι (OData REST API)	☑ Ναι
UI εφαρμογής μίας σελίδας (SPA)	☑ Ναι (React SPA)	☑ Ναι (Ember/React)	☑ Ναι (UCI SPA)	☑ Ναι
Διεπαφή χρήστη				
Πίνακες ελέγχου βάσει ρόλου	☑ Dashboards με gadgets	☑ Dashboards ανά ρόλο	☑ Power BI / role dashboards	☑ Dashboards ανά ρόλο
Προηγμένο φιλτράρισμα και αναζήτηση	☑ JQL	☑ Προηγμένη αναζήτηση	☑ Προηγμένα queries (OData)	☑ Βασικά φίλτρα, saved views
Προηγμένες λειτουργίες				
Ενσωμάτωση email	☑ Email-to-request	☑ Email-to-ticket	☑ Email sync	☑ Tickets από email
Διαχείριση SLA	☑ Διαχείριση SLA	☑ Διαχείριση SLA	☑ Διαχείριση SLA	☑ Βασικό SLA στο Service Hub
Γνωσιακή Βάση/Πύλη αυτοεξυπηρέτησης	☑ Confluence KB integration	☑ Help Center	☑ Knowledge articles	☑ Knowledge Base + portal

Όπως δείχνει η ανάλυση στον Πίνακας 2.1, υπάρχει σαφής συμφωνία σχετικά με τα βασικά χαρακτηριστικά που απαιτούνται για ένα ισχυρό σύστημα διαχείρισης αιτημάτων υποστήριξης. Η έρευνα αυτή επηρέασε άμεσα τις λειτουργικές απαιτήσεις της εφαρμογής.

Ωστόσο, με βάση αυτή την ανάλυση, ελήφθησαν αρκετές αποφάσεις για να διαφοροποιηθεί το προτεινόμενο σύστημα και να ευθυγραμμιστεί με τις σύγχρονες, επεκτάσιμες πρακτικές ανάπτυξης ιστού.

Πρώτον, επιλέχθηκε ένας σχεδιασμός API-first ως βασική αρχιτεκτονική αρχή. Ενώ οι παραδοσιακές επιχειρηματικές πλατφόρμες συχνά έχουν τη λογική τους στενά συνδεδεμένη με μια διεπαφή χρήστη που αποδίδεται από διακομιστή, αυτό το σύστημα σχεδιάστηκε γύρω από ένα ολοκληρωμένο RESTful API. Κατά συνέπεια, η διεπαφή χρήστη αναπτύχθηκε ως μια εντελώς ξεχωριστή εφαρμογή μίας σελίδας (SPA) που χρησιμοποιεί αυτό το API. Αυτή η αποσυνδεδεμένη προσέγγιση παρέχει σημαντική ευελιξία για μελλοντική ανάπτυξη, όπως η δημιουργία μιας εφαρμογής για κινητά ή ενσωματώσεις τρίτων, οι οποίες θα χρησιμοποιούν όλες το ίδιο βασικό API.

Δεύτερον, για να εξασφαλιστεί υψηλότερο επίπεδο σταθερότητας του συστήματος και ακεραιότητας των δεδομένων, ορίστηκε ένας αποσυνδεδεμένος, event-driven μηχανισμός ελέγχου. Αυτή η απόφαση ελήφθη για να αντιμετωπιστούν προληπτικά τα πιθανά προβλήματα ταυτόχρονης πρόσβασης που εντοπίζονται σε πιο μονολιθικά σχέδια ελέγχου, δίνοντας προτεραιότητα στην ασφάλεια των συναλλαγών και τη συντηρησιμότητα.

2.2. Ρόλοι Χρηστών (Actors)

Μετά την ανάλυση των απαιτήσεων, σχεδιάστηκε ένα μοντέλο **Role-Based Access Control (RBAC)** για να εξυπηρετήσει το διαφορετικό σύνολο χρηστών που προσδιορίστηκε από την ανάλυση. Η διαδικασία σχεδιασμού περιλάμβανε την ανάλυση των διακριτών ευθυνών και των αναγκών πληροφόρησης κάθε ομάδας ενδιαφερομένων σε μια τυπική ροή εργασιών υποστήριξης.

Αυτή η ανάλυση οδήγησε στον ορισμό τεσσάρων διακριτών ρόλων, ή «ηθοποιών» (actors). Κάθε ρόλος σχεδιάστηκε με μια συγκεκριμένη λειτουργία και ένα αντίστοιχο σύνολο δικαιωμάτων, δημιουργώντας μια ιεραρχία πρόσβασης από τον εξωτερικό πελάτη έως τον διαχειριστή του συστήματος. Αυτός ο διαχωρισμός ρόλων είναι θεμελιώδης για την ασφάλεια της εφαρμογής και τη διαχείριση της ροής εργασίας.

Ο ρόλος **ADMIN** σχεδιάστηκε για να αντιπροσωπεύει τον διαχειριστή του συστήματος, ο οποίος είναι υπεύθυνος για τη συνολική διαμόρφωση και την καλή λειτουργία της εφαρμογής. Ο κύριος στόχος αυτού του ρόλου είναι η βασική ρύθμιση της πλατφόρμας και όχι η καθημερινή επίλυση αιτημάτων. Οι βασικές ευθύνες που ορίζονται για αυτόν τον ρόλο περιλαμβάνουν την πλήρη διαχείριση των χρηστών, τη διαμόρφωση των ρυθμίσεων σε όλο το σύστημα και τη διαχείριση των custom οντοτήτων.

Ο ρόλος **SUPERVISOR** σχεδιάστηκε για να αντιπροσωπεύει έναν επικεφαλής ομάδας, υπεύθυνο για τη διαχείριση της ροής εργασίας και της λειτουργικής αποδοτικότητας της ομάδας υποστήριξης. Οι κύριες ευθύνες που ανατέθηκαν σε αυτόν τον ρόλο είναι η ταξινόμηση και η διανομή των εισερχόμενων αιτημάτων υποστήριξης. Σε αυτόν τον ρόλο δόθηκε πλήρης ορατότητα σε όλα τα αιτήματα για να διευκολυνθεί ο ποιοτικός έλεγχος, και ο πίνακας ελέγχου του σχεδιάστηκε για να εμφανίζει μετρήσεις απόδοσης σε επίπεδο ομάδας.

Ο ρόλος **SUPPORT_AGENT** αντιπροσωπεύει τον κύριο εσωτερικό χρήστη του συστήματος, υπεύθυνο για την πρακτική εργασία της επίλυσης των προβλημάτων των πελατών. Ο σχεδιασμός της διεπαφής και των δικαιωμάτων αυτού του ρόλου επικεντρώνεται εξ ολοκλήρου στο προσωπικό φόρτο εργασίας του. Οι βασικές δυνατότητες που ορίζονται για αυτόν τον ρόλο περιλαμβάνουν την ανάληψη μη ανατεθειμένων εισιτηρίων, την επικοινωνία με τους πελάτες μέσω δημόσιων απαντήσεων και τη συνεργασία με συναδέλφους μέσω ιδιωτικών εσωτερικών σημειώσεων.

Ο **CUSTOMER** είναι ο εξωτερικός τελικός χρήστης που αναζητά υποστήριξη. Ο σχεδιασμός αυτού του ρόλου έδωσε προτεραιότητα στην ασφάλεια και την προστασία των δεδομένων πάνω από όλα. Τα δικαιώματά του περιορίστηκαν αυστηρά, ώστε να διασφαλιστεί ότι ένας πελάτης μπορεί να δημιουργεί εισιτήρια μόνο για τον εαυτό του και να βλέπει μόνο τα δικά του δεδομένα, χωρίς να έχει ορατότητα στις δραστηριότητες άλλων χρηστών ή στις εσωτερικές λειτουργίες της ομάδας υποστήριξης.

2.3. Περιπτώσεις Χρήσης (Use Cases)

Για τον τελικό καθορισμό των λειτουργικών απαιτήσεων και των αλληλεπιδράσεων μεταξύ των φορέων και του συστήματος, αναπτύχθηκε μια σειρά λεπτομερών περιπτώσεων χρήσης. Κάθε περίπτωση χρήσης περιγράφει έναν συγκεκριμένο στόχο που μπορεί να επιτύχει ένας φορέας χρησιμοποιώντας την εφαρμογή. Η δομή αυτών των περιπτώσεων χρήσης βασίζεται στο πρότυπο

«fully dressed», ένα ολοκληρωμένο format για την λεπτομερή περιγραφή των αλληλεπιδράσεων μεταξύ χρήστη και συστήματος [11].

Αυτή η επίσημη προσέγγιση διασφαλίζει ότι όλες οι απαιτήσεις καταγράφονται με σαφήνεια, παρέχοντας ένα σαφές σχέδιο για τις φάσεις υλοποίησης και δομής του συστήματος. Οι ακόλουθες ενότητες περιγράφουν λεπτομερώς τις κύριες περιπτώσεις χρήσης για καθέναν από τους τέσσερις καθορισμένους ρόλους.

2.3.1. Περιπτώσεις Χρήσης Διαχειριστή (Admin)

- **Διαχείριση Ρόλων και Στοιχείων Χρήστη:**

Όνομα Περίπτωσης: Διαχείριση Ρόλων και Στοιχείων Χρήστη

Ηθοποιός: ADMIN (Διαχειριστής)

Πεδίο Εφαρμογής: Η Εφαρμογή

Επίπεδο: Στόχος Χρήστη

Προϋποθέσεις: Ο Διαχειριστής είναι αυθεντικοποιημένος και συνδεδεμένος στο σύστημα.

Εγγύηση Επιτυχίας: Τα χαρακτηριστικά του χρήστη-στόχου (όνομα, επώνυμο, ρόλος) αποθηκεύονται επιτυχώς στη βάση δεδομένων και εμφανίζεται ειδοποίηση επιτυχίας.

Κύριο Σενάριο Επιτυχίας**:**

1. Ο Διαχειριστής πλοηγείται στη σελίδα "Users" κάτω από το “Admin Panel” από την μπάρα στα αριστερά ή από την μπάρα πλοήγησης σε μικρές συσκευές.
2. Το σύστημα εμφανίζει μια σελιδοποιημένη λίστα όλων των χρηστών με δυνατότητες φιλτραρίσματος.
3. Ο Διαχειριστής εντοπίζει τον χρήστη-στόχο και επιλέγει το κουμπί "Edit".
4. Το σύστημα εμφανίζει το παράθυρο "Edit User", προ-συμπληρωμένο με τα τρέχοντα στοιχεία του χρήστη.
5. Ο Διαχειριστής τροποποιεί το όνομα του χρήστη ή/και επιλέγει έναν νέο ρόλο από το αναπτυσσόμενο μενού και πατάει "Save Changes".
6. Το σύστημα κλείνει το παράθυρο και ανανεώνει τη λίστα χρηστών για να εμφανίσει τα νέα δεδομένα.

Επεκτάσεις: 5α. Ο Διαχειριστής πατάει "Cancel": Η περίπτωση χρήσης τερματίζεται χωρίς να αποθηκευτεί καμία αλλαγή.

- **Παραμετροποίηση Ρυθμίσεων Εφαρμογής**

Όνομα Περίπτωσης: Διαχείριση Κατηγοριών Αιτημάτων

Ηθοποιός: ADMIN (Διαχειριστής)

Πεδίο Εφαρμογής: Η Εφαρμογή

Επίπεδο: Στόχος Χρήστη

Προϋποθέσεις: Ο Διαχειριστής είναι αυθεντικοποιημένος και συνδεδεμένος στο σύστημα.

Εγγύηση Επιτυχίας: Οι νέες ρυθμίσεις της εφαρμογής αποθηκεύονται επιτυχώς στη βάση δεδομένων και εμφανίζεται ειδοποίηση επιτυχίας.

Κύριο Σενάριο Επιτυχίας**:**

1. Ο Διαχειριστής πλοηγείται στη σελίδα "Settings" κάτω από το dropdown με το όνομα του από την μπάρα πλοήγησης.
2. Ο Διαχειριστής διαλέγει την καρτέλα “Application”.
3. Το σύστημα εμφανίζει τα τρέχοντα πεδία για το όνομα της εφαρμογής και την επιλογή λογότυπου
4. **Για Αλλαγή Ονόματος:** Ο Διαχειριστής πληκτρολογεί ένα νέο όνομα στο πεδίο και πατάει "Save Name". Το νέο όνομα εμφανίζεται μετά την ανανέωση της σελίδας.
5. **Για Αλλαγή Λογότυπου:** Ο Διαχειριστής επιλέγει ένα αρχείο εικόνας και πατάει "Upload Logo". Το νέο λογότυπο εμφανίζεται μετά την ανανέωση της σελίδας.

Επεκτάσεις: 5α. Ο χρήστης ανεβάζει αρχείο λάθος τύπου ή μεγέθους: εμφανίζεται μια ειδοποίηση σφάλματος και δεν γίνεται αλλαγή του λογότυπου.

Εκτός από τις κύριες λειτουργίες που περιγράφονται παραπάνω με λεπτομέρεια, ο διαχειριστής είναι υπεύθυνος για διάφορες άλλες κρίσιμες εργασίες διαχείρισης, οι οποίες ακολουθούν παρόμοια μοτίβα αλληλεπίδρασης, όπως η προβολή μιας λίστας, το άνοιγμα ενός παραθύρου και η υποβολή μιας φόρμας. Αυτές περιλαμβάνουν:

- **Δημιουργία νέου χρήστη:** Ο διαχειριστής μπορεί να ξεκινήσει τη διαδικασία δημιουργίας χρήστη από τη σελίδα «Users». Αυτό ανοίγει ένα ειδικό παράθυρο όπου μπορεί να εισαγάγει το όνομα, το επώνυμο, το email και τον αρχικό κωδικό πρόσβασης του νέου χρήστη, καθώς και να του εκχωρήσει έναν ρόλο από μια προκαθορισμένη λίστα.
- **Διαγραφή χρήστη:** Από τη λίστα χρηστών, ο διαχειριστής μπορεί να ενεργοποιήσει τη διαγραφή ενός χρήστη. Το σύστημα εμφανίζει ένα παράθυρο επιβεβαίωσης για να αποτρέψει την τυχαία απώλεια δεδομένων. Μετά την επιβεβαίωση, ο χρήστης και όλα τα σχετικά με αυτόν διακριτικά συνεδρίας διαγράφονται μόνιμα από το σύστημα.
- **Διαχείριση κατηγοριών:** Ο διαχειριστής έχει πλήρη έλεγχο του κύκλου ζωής των κατηγοριών εισιτηρίων μέσω της σελίδας «Διαχείριση κατηγοριών». Αυτό περιλαμβάνει:
    - **Δημιουργία** νέων κατηγοριών.
    - **Μετονομασία** υπαρχουσών κατηγοριών.
    - **Απενεργοποίηση (προσωρινή διαγραφή)** κατηγοριών που δεν χρησιμοποιούνται πλέον.
    - **Επανενεργοποίηση** κατηγοριών που είχαν απενεργοποιηθεί προηγουμένως.
- **Προβολή πλήρους ιστορικού εισιτηρίων:** Αν και δεν είναι ο κύριος παράγοντας στην επίλυση εισιτηρίων, ο διαχειριστής έχει το υψηλότερο επίπεδο προνομίων, το οποίο του επιτρέπει να προβάλλει και να έχει πρόσβαση σε όλες τις λεπτομέρειες και το ιστορικό ελέγχου οποιουδήποτε εισιτηρίου υποστήριξης στο σύστημα για σκοπούς εποπτείας ή αντιμετώπισης προβλημάτων.

2.3.2. Περιπτώσεις Χρήσης Επόπτη (Supervisor)

- **Ανάθεση Νέου Αιτήματος:**

Όνομα Περίπτωσης: Διαχείριση Ρόλων και Στοιχείων Χρήστη

Ηθοποιός: SUPERVISOR (Επόπτης)

Πεδίο Εφαρμογής: Η Εφαρμογή

Επίπεδο: Στόχος Χρήστη

Προϋποθέσεις: Ο Υπεύθυνος είναι αυθεντικοποιημένος. Υπάρχει τουλάχιστον ένα μη ανατεθειμένο αίτημα με κατάσταση NEW.

Εγγύηση Επιτυχίας: Το αίτημα ανατίθεται επιτυχώς σε έναν Εκπρόσωπο Υποστήριξης, η κατάστασή του ενημερώνεται σε OPEN, και δημιουργείται μια εγγραφή στο αρχείο καταγραφής.

Κύριο Σενάριο Επιτυχίας**:**

1. Ο Επόπτη πλοηγείται στη σελίδα "Tickets" από την μπάρα στα αριστερά ή από την μπάρα πλοήγησης σε μικρές συσκευές.
2. Το σύστημα εμφανίζει μια σελιδοποιημένη λίστα όλων των αιτήσεων με δυνατότητες φιλτραρίσματος.
3. Ο Επόπτη εντοπίζει την αίτηση-στόχο και την επιλέγει κάνοντας κλικ πάνω στο αναγνωριστικό της νούμερο στην στήλη ID.
4. Το σύστημα εμφανίζει την σελίδα του αιτήματος.
5. Ο Επόπτη επιλέγει από την μπάρα εργαλείων το κουμπί “Assign”.
6. Το σύστημα εμφανίζει το παράθυρο "Assign Ticket".
7. Ο Επόπτη πληκτρολογεί ένα όνομα για να αναζητήσει και επιλέγει έναν Υπάλληλο Υποστήριξης (Support Agent) από την λίστα και πατάει το κουμπί "Assign".
8. Το σύστημα εμφανίζει ειδοποίηση επιτυχίας, κλείνει το παράθυρο και ανανεώνει τη σελίδα του αιτήματος για να εμφανιστεί το νέο όνομα στο πεδίο “Assignee”

Επεκτάσεις: 7α. Ο Επόπτη πατάει "Cancel": Η περίπτωση χρήσης τερματίζεται χωρίς να αποθηκευτεί καμία αλλαγή.

Πέρα από την απευθείας ανάθεση, η ροή εργασίας του Επόπτη υποστηρίζεται από τις ακόλουθες βασικές δυνατότητες:

- **Παρακολούθηση Όλων των Ουρών Αιτημάτων:** Σε αντίθεση με έναν απλό Εκπρόσωπο, ο Επόπτης μπορεί να προβάλει και να φιλτράρει την πλήρη λίστα όλων των αιτημάτων υποστήριξης στο σύστημα, ανεξαρτήτως του ποιος είναι ο υπεύθυνος. Αυτό του επιτρέπει να παρακολουθεί τον συνολικό φόρτο εργασίας, να εντοπίζει σημεία συμφόρησης και να παρακολουθεί την πρόοδο των αιτημάτων υψηλής προτεραιότητας.
- **Επανάθεση Αιτημάτων:** Ο Επόπτης μπορεί να αλλάξει τον υπεύθυνο ενός ήδη ανατεθειμένου αιτήματος. Αυτή είναι μια κρίσιμη λειτουργία για την εξισορρόπηση του φόρτου εργασίας, σε περίπτωση που ένας εκπρόσωπος είναι υπερφορτωμένος ή βρίσκεται σε άδεια.
- **Προβολή Dashboard Απόδοσης Ομάδας:** Το Dashboard του Επόπτη παρέχει μια υψηλού επιπέδου επισκόπηση ολόκληρης της ουράς υποστήριξης, συμπεριλαμβανομένων στατιστικών για τα μη ανατεθειμένα αιτήματα, γράφημα των αιτημάτων ανά κατάσταση, και συνολικών ποσοστών επίλυσης.
- **Πρόσβαση στο Ιστορικό Καταγραφών (Auditing):** Ο Επόπτης, ως προνομιούχος ρόλος, έχει πρόσβαση στην καρτέλα "Ιστορικό" στη σελίδα λεπτομερειών ενός αιτήματος, επιτρέποντάς του να εξετάσει το πλήρες και αμετάβλητο ιστορικό όλων των αλλαγών που έχουν γίνει σε ένα αίτημα.

2.3.3. Περιπτώσεις Χρήσης Εκπροσώπου Υποστήριξης (Support Agent)

- **Προσθήκη Σχολίου σε Αίτημα:**

Όνομα Περίπτωσης: Προσθήκη Σχολίου σε Αίτημα

Ηθοποιός: SUPPORT_AGENT (Εκπρόσωπος Υποστήριξης)

Πεδίο Εφαρμογής: Η Εφαρμογή

Επίπεδο: Στόχος Χρήστη

Προϋποθέσεις: Ο Εκπρόσωπος είναι αυθεντικοποιημένος και προβάλλει ένα αίτημα στο οποίο έχει δικαίωμα πρόσβασης (είτε είναι δικό του είτε μη ανατεθειμένο).

Εγγύηση Επιτυχίας: Ένα νέο σχόλιο αποθηκεύεται στη βάση δεδομένων, και η κατάσταση του αιτήματος ενημερώνεται αυτόματα εάν απαιτείται.

Κύριο Σενάριο Επιτυχίας**:**

1. Ο Εκπρόσωπος πλοηγείται στη σελίδα "Tickets" από την μπάρα στα αριστερά ή από την μπάρα πλοήγησης σε μικρές συσκευές.
2. Το σύστημα εμφανίζει μια σελιδοποιημένη λίστα όλων των αιτήσεων που είναι είτε ανατεθειμένες στον επόπτη είτε δεν έχουν κάποιον assignee με δυνατότητες φιλτραρίσματος.
3. Ο Εκπρόσωπος εντοπίζει την αίτηση-στόχο και την επιλέγει κάνοντας κλικ πάνω στο αναγνωριστικό της νούμερο στην στήλη ID.
4. Το σύστημα εμφανίζει την σελίδα του αιτήματος.
5. Ο Εκπρόσωπος πληκτρολογεί την απάντηση του σε σχόλιο στο “Add Your Reply” και διαλέγει εάν είναι εσωτερικό μήνυμα ή όχι από το κουμπί “Make this an internal note”.
6. Το σύστημα επιστρέφει ειδοποίηση επιτυχίας και ανανεώνει την σελίδα του αιτήματος για να εμφανιστεί το νέο σχόλιο (με ειδική σήμανση αν είναι εσωτερικό).

Επεκτάσεις: 5α. Το αίτημα έχει κλείσει και δεν μπορεί να λάβει περαιτέρω σχόλια οπότε η περίπτωση χρήσης τερματίζεται.

Άλλες Βασικές Περιπτώσεις Χρήσης:

- Ανάληψη Μη Ανατεθειμένου Αιτήματος (Claim): Ο Εκπρόσωπος μπορεί να περιηγηθεί στην ουρά των μη ανατεθειμένων αιτημάτων. Από τη σελίδα λεπτομερειών ενός τέτοιου αιτήματος, μπορεί να πατήσει το κουμπί "Ανάληψη" (Claim). Αυτή η ενέργεια αναθέτει αυτόματα το αίτημα στον εαυτό του και αλλάζει την κατάστασή του σε OPEN.
- Επίλυση Αιτήματος: Όταν ο Εκπρόσωπος θεωρεί ότι έχει δώσει οριστική λύση στο πρόβλημα του πελάτη, μπορεί να πατήσει το κουμπί "Επίλυση" (Mark as Resolved). Αυτό αλλάζει την κατάσταση του αιτήματος σε RESOLVED και θέτει αυτόματα τη χρονική σήμανση resolvedAt, κινώντας το αίτημα προς το τελικό στάδιο της αρχειοθέτησης.
- Ενημέρωση Ιδιοτήτων Αιτήματος: Ο Εκπρόσωπος έχει τη δυνατότητα να τροποποιήσει τις ιδιότητες των αιτημάτων που διαχειρίζεται (π.χ. αλλαγή Προτεραιότητας, Κατηγορίας) μέσω του παραθύρου "Ενημέρωση Ιδιοτήτων", ώστε να αντικατοπτρίζουν με ακρίβεια την τρέχουσα φύση του ζητήματος.

2.3.4. Περιπτώσεις Χρήσης Πελάτη (Customer)

- **Δημιουργία Νέου Αιτήματος Υποστήριξης:**

Όνομα Περίπτωσης: Δημιουργία Νέου Αιτήματος Υποστήριξης

Ηθοποιός: CUSTOMER (Πελάτης)

Πεδίο Εφαρμογής: Η Εφαρμογή

Επίπεδο: Στόχος Χρήστη

Προϋποθέσεις: Ο Πελάτης είναι αυθεντικοποιημένος και συνδεδεμένος στο σύστημα.

Εγγύηση Επιτυχίας: Ένα νέο SupportTicket δημιουργείται επιτυχώς με κατάσταση NEW, τυχόν συνημμένα αποθηκεύονται και η ενέργεια καταγράφεται στο σύστημα.

Κύριο Σενάριο Επιτυχίας**:**

1. Ο Πελάτης πατάει το κουμπί "Create New Ticket" από το προσωπικό του Dashboard.
2. Το σύστημα τον πλοηγεί στην ειδική φόρμα δημιουργίας αιτήματος.
3. Ο Πελάτης συμπληρώνει όλα τα απαιτούμενα πεδία (Θέμα, Λεπτομερής Περιγραφή, Κατηγορία, Προτεραιότητα)
4. (Προαιρετικά) Ο Πελάτης χρησιμοποιεί το πεδίο εισαγωγής αρχείων για να επισυνάψει ένα ή περισσότερα σχετικά αρχεία (π.χ. screenshots, log files). Η διεπαφή χρήστη εμφανίζει μια λίστα με τα επιλεγμένα αρχεία, παρέχοντας τη δυνατότητα αφαίρεσης πριν την υποβολή.
5. Ο Πελάτης πατάει το κουμπί "Submit Ticket".
6. Ο χρήστης ανακατευθύνεται αυτόματα στη σελίδα λεπτομερειών του αιτήματος που μόλις δημιούργησε.

Επεκτάσεις: 5α. Ένα από τα αρχεία αποτυγχάνει την επικύρωση (π.χ. υπερβαίνει το μέγιστο επιτρεπτό μέγεθος): Το API επιστρέφει σφάλμα 400 Bad Request. Η συναλλαγή του backend αποτυγχάνει πλήρως (rollback), και δεν αποθηκεύεται ούτε το αίτημα ούτε κανένα από τα αρχεία. Ο χρήστης ενημερώνεται με ένα μήνυμα σφάλματος.

Άλλες Βασικές Περιπτώσεις Χρήσης:

- **Προβολή Ιστορικού Αιτημάτων:** Ο Πελάτης μπορεί να πλοηγηθεί στη σελίδα "View All My Tickets" για να δει μια πλήρη, σελιδοποιημένη λίστα όλων των αιτημάτων που έχει υποβάλει στο παρελθόν, μαζί με τη βασική τους κατάσταση.
- **Προσθήκη Απάντησης σε Αίτημα:** Ο Πελάτης μπορεί να προβάλει τις λεπτομέρειες ενός ενεργού αιτήματος και να προσθέσει μια νέα απάντηση ή διευκρίνιση. Εάν το αίτημα είχε κατάσταση WAITING_CUSTOMER ή RESOLVED, η υποβολή της απάντησης αλλάζει αυτόματα την κατάστασή του σε IN_PROGRESS, ειδοποιώντας την ομάδα υποστήριξης ότι απαιτείται η προσοχή τους.

2.4. Τεχνική Ανάλυση

Η επιλογή των τεχνολογιών για το έργο ήταν μια σκόπιμη διαδικασία, καθοδηγούμενη από τις λειτουργικές και μη λειτουργικές απαιτήσεις που είχαν καθοριστεί προηγουμένως. Ο στόχος ήταν να επιλεγούν ισχυρές και ευρέως χρησιμοποιημένες τεχνολογίες που θα επέτρεπαν την ανάπτυξη μιας ασφαλούς, επεκτάσιμης και συντηρήσιμης full stack εφαρμογής. Η αρχιτεκτονική αποτελείται από έναν διακομιστή backend, έναν πελάτη frontend, μια βάση δεδομένων και έναν μηχανισμό ασφαλείας, καθένας από τους οποίους υλοποιείται με μια συγκεκριμένη, δικαιολογημένη τεχνολογία.

2.4.1. Διακομιστής Backend

Για την υλοποίηση της επιχειρηματικής λογικής από την πλευρά του διακομιστή και του RESTful API, επιλέχθηκε το **Spring Boot framework**(Java). Το Spring Boot είναι ένα κορυφαίο πλαίσιο για τη

δημιουργία αυτόνομων εφαρμογών επιχειρησιακού επιπέδου. Τα κύρια πλεονεκτήματά του για αυτό το project περιλαμβάνουν:

- **Ταχεία ανάπτυξη:** Η φιλοσοφία «συμβατική ρύθμιση αντί για διαμόρφωση» και οι εκτεταμένες δυνατότητες αυτόματης διαμόρφωσης μειώνουν σημαντικά τον boilerplate κώδικα και επιταχύνουν τη διαδικασία ανάπτυξης.
- **Ολοκληρωμένο οικοσύστημα:** Παρέχει ισχυρά, προκατασκευασμένα modules για κρίσιμες λειτουργίες, με πιο αξιοσημείωτα το **Spring Security** για την αυθεντικοποίηση και την εξουσιοδότηση και το **Spring Data JPA** για την αλληλεπίδραση με τη βάση δεδομένων.
- **Έγχυση εξαρτήσεων (dependency injection):** Το βασικό του Inversion of Control (IoC) container απλοποιεί τη διαχείριση των στοιχείων της εφαρμογής και προάγει μια χαλαρά συνδεδεμένη, αρθρωτή αρχιτεκτονική.

2.4.2. Πελάτης Frontend

Το επίπεδο παρουσίασης από την πλευρά του πελάτη αναπτύχθηκε ως **εφαρμογή μίας σελίδας (SPA)** χρησιμοποιώντας τη βιβλιοθήκη **React**. Το React επιλέχθηκε για την αρχιτεκτονική του που βασίζεται σε components, η οποία επιτρέπει τη δημιουργία της διεπαφής χρήστη ως σύνθεση μικρών, επαναχρησιμοποιήσιμων και αυτόνομων τμημάτων. Αυτή η προσέγγιση βελτιώνει τη συντηρησιμότητα και την επεκτασιμότητα του κώδικα. Η χρήση ενός εικονικού DOM παρέχει υψηλή απόδοση ελαχιστοποιώντας τις άμεσες παρεμβάσεις στο πραγματικό DOM, με αποτέλεσμα μια γρήγορη και ομαλή εμπειρία χρήστη [12].

Για να επιταχυνθεί η ανάπτυξη ενός επαγγελματικού και πλήρως ανταποκρινόμενου περιβάλλοντος εργασίας χρήστη, η βιβλιοθήκη **React Bootstrap** ενσωματώθηκε στο project [13]. Αυτή η βιβλιοθήκη παρέχει ένα πλήρες σύνολο προκατασκευασμένων στοιχείων React που εφαρμόζουν το δημοφιλές σύστημα σχεδιασμού Bootstrap. Αντί να γράφει προσαρμοσμένο CSS για τυπικά στοιχεία UI όπως modals, πίνακες και κουμπιά, αυτό το project αξιοποιεί τα στοιχεία του React Bootstrap. Αυτό όχι μόνο επιτάχυνε την ανάπτυξη, αλλά και εξασφάλισε ότι ολόκληρη η εφαρμογή ακολουθεί μια συνεπή γλώσσα σχεδιασμού και είναι ανταποκρινόμενη, προσαρμόζοντας σε ένα ευρύ φάσμα συσκευών, από κινητά τηλέφωνα έως υπολογιστές.

2.4.3. Βάση Δεδομένων

Για το επίπεδο δεδομένων, επιλέχθηκε μια σχεσιακή βάση δεδομένων **MySQL**. Ένα σχεσιακό μοντέλο θεωρήθηκε απαραίτητο για την επιβολή της αυστηρής ακεραιότητας των δεδομένων και των πολύπλοκων σχέσεων που απαιτεί το σύστημα, όπως οι συνδέσεις μεταξύ χρηστών, εισιτηρίων και σχολίων. Η χρήση μιας σχεσιακής βάσης δεδομένων εγγυάται τη συμμόρφωση με το πρότυπο **ACID (Atomicity, Consistency, Isolation, Durability)** για όλες τις συναλλαγές [14].

Η αλληλεπίδραση με τη βάση δεδομένων διαχειρίζεται μέσω του **Java Persistence API (JPA)** με το **Hibernate** ως πάροχο διατήρησης. Αυτό το πλαίσιο αντικειμενικής-σχεσιακής αντιστοίχισης (ORM) αφαιρεί το υποκείμενο SQL, επιτρέποντας στους προγραμματιστές να αλληλεπιδρούν με τη βάση δεδομένων χρησιμοποιώντας γνωστά αντικείμενα και Java annotations. Αυτή η προσέγγιση καθιστά την εφαρμογή πιο φορητή, καθώς μπορεί να μεταφερθεί σε άλλη σχεσιακή βάση δεδομένων με ελάχιστες αλλαγές στον κώδικα και πιο ασφαλή σε ευπάθειες όπως τα SQL injections.

2.4.4. Μηχανισμός αυθεντικοποίησης

Η αυθεντικοποίηση και η διαχείριση της περιόδου λειτουργίας για το stateless RESTful API γίνεται με τη χρήση **JSON Web Tokens (JWT)**. Το JWT είναι ένα **αυτοπεριεχόμενο και ασφαλές μέσο** για τη μεταφορά πληροφοριών μεταξύ δύο μερών σε μορφή JSON, το οποίο είναι βελτιστοποιημένο για χρήση σε περιβάλλοντα ιστού. [15]. Μετά την επιτυχή σύνδεση, ο διακομιστής δημιουργεί ένα υπογεγραμμένο JWT που περιέχει την ταυτότητα και τον ρόλο του χρήστη και το στέλνει στον πελάτη. Στη συνέχεια, ο πελάτης περιλαμβάνει αυτό το token στην κεφαλίδα εξουσιοδότησης κάθε επόμενης αίτησης.

Αυτή η προσέγγιση βάσει token επιλέχθηκε επειδή είναι χωρίς κατάσταση. Ο διακομιστής δεν χρειάζεται να αποθηκεύει πληροφορίες περιόδου λειτουργίας στη μνήμη ή στη βάση δεδομένων για να πιστοποιήσει έναν χρήστη, καθιστώντας το σύστημα εξαιρετικά επεκτάσιμο και ιδανικό για αποσυνδεδεμένες αρχιτεκτονικές.

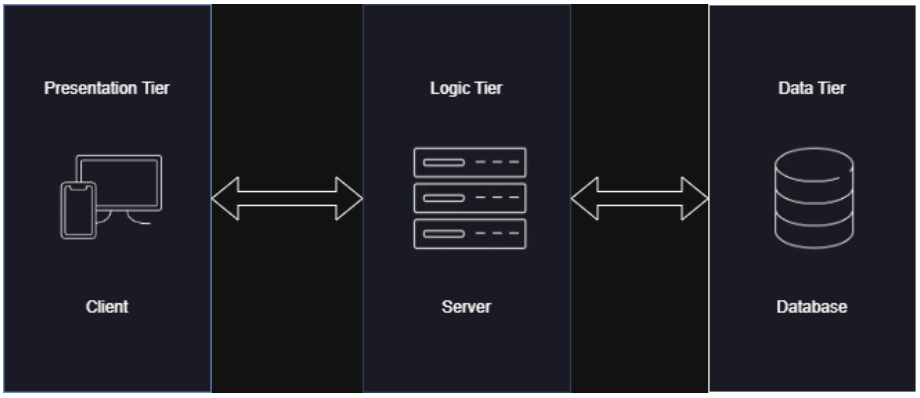
3. Αρχιτεκτονική και Σχεδίαση Βάσης Δεδομένων

3.1. Γενική Αρχιτεκτονική

Η αρχιτεκτονική βάση της εφαρμογής είναι μια αρχιτεκτονική τριών επιπέδων [4]. Αυτό το αποσυνδεδεμένο πρότυπο σχεδιασμού επιλέχθηκε για την επεκτασιμότητα, τη συντηρησιμότητα και τον σαφή διαχωρισμό των αρμοδιοτήτων του. Μια αρχιτεκτονική τριών επιπέδων διαχωρίζει λογικά μια εφαρμογή σε τρία ξεχωριστά, αλληλοσυνδεόμενα επίπεδα:

1. Επίπεδο παρουσίασης (η «προβολή»): Αυτό είναι το επίπεδο που βλέπει ο χρήστης, το οποίο είναι υπεύθυνο για όλη τη διεπαφή χρήστη (user interface) και την αλληλεπίδραση του χρήστη με αυτήν. Σε αυτή την διατριβή, υλοποιείται ως εφαρμογή React Single-Page Application (SPA). Επικοινωνεί με το επιχειρηματικό επίπεδο αποκλειστικά μέσω ενός RESTful API μέσω HTTP.
2. Επίπεδο επιχειρηματικής λογικής (διακομιστής εφαρμογών): Αυτό είναι ο πυρήνας της εφαρμογής, υπεύθυνος για την επιβολή όλων των επιχειρηματικών κανόνων, την επεξεργασία δεδομένων και τη διαχείριση της ασφάλειας. Υλοποιείται με ένα Spring Boot RESTful API. Αυτό το επίπεδο ακολουθεί μια λογική παρόμοια με το Model-View-Controller [16], όπου:
    - Οι ελεγκτές (controllers) χειρίζονται τα εισερχόμενα αιτήματα HTTP.
    - Οι υπηρεσίες (services) περιέχουν την βασική επιχειρηματική λογική.
    - Η «Προβολή» (View) σε αυτό το πλαίσιο είναι τα δεδομένα JSON που αποστέλλονται πίσω στο επίπεδο παρουσίασης.
3. Επίπεδο δεδομένων (διακομιστής βάσης δεδομένων): Αυτό το επίπεδο είναι υπεύθυνο για τη διατήρηση και την ανάκτηση των δεδομένων. Υλοποιείται με μια σχεσιακή βάση δεδομένων MySQL και η πρόσβαση σε αυτό γίνεται μέσω του Java Persistence API (JPA) με Hibernate, το οποίο παρέχει μια αφαίρεση αντικειμενικής σχεσιακής αντιστοίχισης (ORM) πάνω από τη βάση δεδομένων.

Παρακάτω στο Διάγραμμα 3.1 φαίνεται σχηματικά η αρχιτεκτονική που περιγράφτηκε.

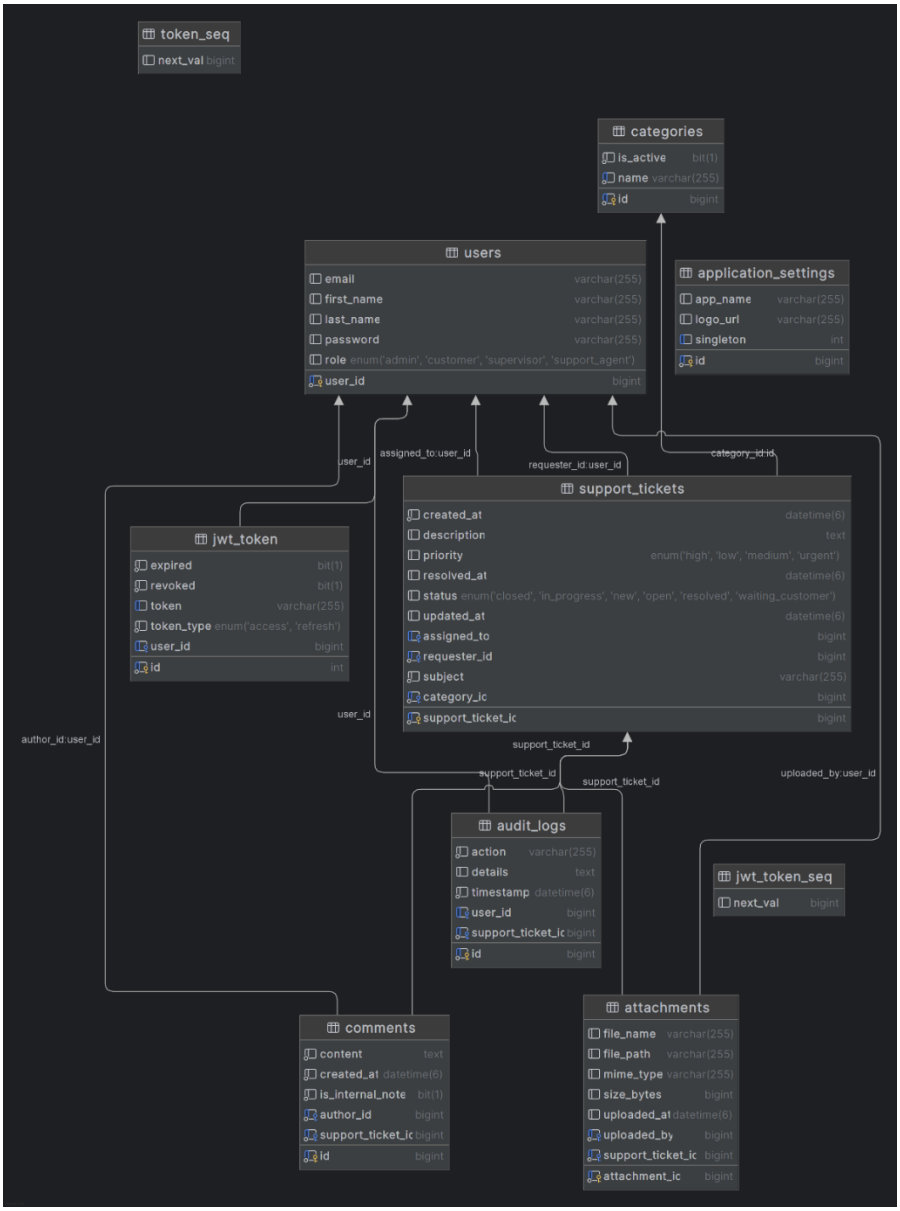


Διάγραμμα 3.1 Αρχιτεκτονική τριών επιπέδων

3.2. Σχεδίαση Σχεσιακής Βάσης Δεδομένων

Το επίπεδο των δεδομένων της εφαρμογής βασίζεται σε ένα μοντέλο σχεσιακής βάσης δεδομένων, το οποίο υλοποιήθηκε χρησιμοποιώντας MySQL. Επιλέχθηκε για να διασφαλιστεί η ακεραιότητα των δεδομένων και να διαχειριστεί τις πολύπλοκες σχέσεις μεταξύ των βασικών οντοτήτων του συστήματος. Το σχήμα της βάσης δεδομένων αναπτύχθηκε χρησιμοποιώντας μια προσέγγιση «code-first», όπου οι οντότητες του Java Persistence API (JPA), που ορίζονται ως σχολιασμένες (με annotation) κλάσεις Java, χρησιμοποιούνται από το Hibernate για την αυτόματη δημιουργία και διαχείριση της δομής της βάσης δεδομένων.

Το πλήρες σχήμα της βάσης δεδομένων, που απεικονίζει όλες τις οντότητες και τις σχέσεις τους, παρουσιάζεται στο Διάγραμμα 3.2.



Διάγραμμα 3.2 Διάγραμμα Σχέσεων-Οντοτήτων (ERD) του Συστήματος

Το σχήμα αποτελείται από επτά βασικές οντότητες, καθεμία από τις οποίες έχει σχεδιαστεί για να εκπληρώνει έναν συγκεκριμένο ρόλο εντός του τομέα της εφαρμογής, όπως περιγράφεται λεπτομερώς παρακάτω.

- Οντότητα χρήστη (users):

Ο πίνακας users είναι κεντρικός για την εφαρμογή και έχει σχεδιαστεί για την αποθήκευση πληροφοριών για όλα τα άτομα που αλληλεπιδρούν με το σύστημα. Ένα βασικό χαρακτηριστικό του σχεδιασμού είναι η ικανότητά του να αντιπροσωπεύει τόσο το εσωτερικό προσωπικό (διαχειριστές, επόπτες, πράκτορες) όσο και τους εξωτερικούς πελάτες μέσω της χρήσης ενός χαρακτηριστικού ρόλου. Αυτό το χαρακτηριστικό αποτελεί τον ακρογωνιαίο λίθο του μοντέλου ελέγχου πρόσβασης βάσει ρόλων (RBAC). Το πεδίο email χρησιμεύει ως μοναδικός αναγνωριστικός κωδικός για σκοπούς πιστοποίησης, ενώ το πεδίο κωδικού πρόσβασης αποθηκεύει τον κωδικό με την μορφή hash αφού έχει περάσει από κρυπτογράφηση.

- Οντότητα κατηγορίας (categories):

Η οντότητα category επιτρέπει τη δυναμική ταξινόμηση των εισιτηρίων υποστήριξης. Αυτό παρέχει στους διαχειριστές την ευελιξία να προσθέτουν, να μετονομάζουν ή να απενεργοποιούν κατηγορίες μέσω του UI χωρίς να απαιτείται να γίνει αλλαγή στον κώδικα. Το boolean is_active ενεργοποιεί έναν μηχανισμό «soft delete», διασφαλίζοντας ότι τα ιστορικά δεδομένα των εισιτηρίων παραμένουν ανέπαφα ακόμη και αν μια κατηγορία απενεργοποιηθεί, διατηρώντας έτσι την αναφορική ακεραιότητα.

- Οντότητα SupportTicket (support_tickets):

Αυτή είναι η κύρια οντότητα συναλλαγών, που αντιπροσωπεύει ένα μεμονωμένο αίτημα υποστήριξης. Συνδέει πολλές άλλες οντότητες μέσω σχέσεων πολλών προς ένα: πρέπει να έχει έναν αιτούντα (users), μία κατηγορία (categories) και μπορεί προαιρετικά να έχει έναν υπεύθυνο (users). Τα πεδία κατάστασης και προτεραιότητας υλοποιούνται ως Enum πεδία, καθώς αντιπροσωπεύουν ένα σταθερό σύνολο εσωτερικών καταστάσεων επιχειρηματικής λογικής.

- Οντότητα σχολίων (comments):

Η οντότητα σχολίων έχει σχεδιαστεί για να είναι ένα αμετάβλητο αρχείο της συνομιλίας που σχετίζεται με ένα εισιτήριο υποστήριξης. Κάθε σχόλιο συνδέεται με ένα μόνο εισιτήριο υποστήριξης και έναν μόνο συντάκτη (έναν χρήστη). Ένα κρίσιμο χαρακτηριστικό είναι το flag is_internal_note, το οποίο διαχωρίζει τις δημόσιες απαντήσεις που απευθύνονται στους πελάτες από τις ιδιωτικές σημειώσεις που προορίζονται μόνο για το εσωτερικό προσωπικό, παρέχοντας έτσι ένα ασφαλές κανάλι για τη συνεργασία της ομάδας.

- Οντότητα συνημμένων/εγγράφων (attachments):

Αυτή η οντότητα αποθηκεύει μεταδεδομένα για αρχεία που έχουν μεταφορτωθεί στο σύστημα. Ενώ τα φυσικά αρχεία αποθηκεύονται στο τοπικό σύστημα αρχείων του διακομιστή, αυτή η εγγραφή βάσης δεδομένων παρέχει τον απαραίτητο σύνδεσμο και το path στον διακομιστή. Περιέχει πληροφορίες όπως το αρχικό file_name, το mime_type και το μοναδικό file_path που δημιουργείται από το σύστημα. Κάθε συνημμένο συνδέεται με ένα μόνο support_ticket.

- Οντότητα AuditLog (audit_logs):

Αυτή η οντότητα παρέχει ένα λεπτομερές, αμετάβλητο αρχείο καταγραφής όλων των σημαντικών αλλαγών κατάστασης σε ένα SupportTicket, αποτελώντας τη βάση των δυνατοτήτων ελέγχου του συστήματος. Κάθε καταχώριση στο αρχείο καταγραφής συνδέεται με ένα support_ticket και, προαιρετικά, με έναν actor (χρήστη) που πραγματοποίησε την αλλαγή. Το ξένο κλειδί actor είναι

nullable για να λαμβάνει υπόψη τα αυτοματοποιημένα συμβάντα του συστήματος, όπως το αυτόματο κλείσιμο των επιλυμένων εισιτηρίων.

- Οντότητα ApplicationSettings (application_settings):

Πρόκειται για έναν μοναδικό πίνακα singleton, σχεδιασμένο να περιέχει μόνο μία σειρά από καθολικές ρυθμίσεις που μπορούν να διαμορφωθούν από τον διαχειριστή. Αυτή τη στιγμή αποθηκεύει το δυναμικό app_name της εφαρμογής και τη διαδρομή προς το προσαρμοσμένο logo_url, επιτρέποντας την εύκολη επωνυμία της εφαρμογής χωρίς αλλαγές στον κώδικα.

Αυτό το σχεσιακό μοντέλο, με τις σαφώς καθορισμένες οντότητες και σχέσεις του, παρέχει μια ισχυρή και επεκτάσιμη βάση για το επίπεδο διατήρησης δεδομένων της εφαρμογής.

3.3. Σχεδίαση του RESTful API

Το επίπεδο επικοινωνίας μεταξύ του React frontend και του Spring Boot backend είναι ένα Representational State Transfer (REST) API. Η αρχιτεκτονική REST επιλέχθηκε καθώς αποτελεί προγραμματιστικό πρότυπο για τη δημιουργία επεκτάσιμων, stateless και συντηρήσιμων υπηρεσιών web. Οι αρχές του εξασφαλίζουν μια σαφή και προβλέψιμη συμπεριφορά μεταξύ του πελάτη και του διακομιστή, η οποία είναι απαραίτητη για μια αποσυνδεδεμένη, τριεπίπεδη αρχιτεκτονική που έχει χρησιμοποιηθεί στην περίπτωση της παρούσας εργασίας.

3.3.1. Δομή Uniform Resource Identifier (URI) και Πόρων (Resources)

Οι κύριοι πόροι του συστήματος εκτίθενται μέσω μιας ιεραρχικής δομής URI, με έκδοση κάτω από τη βασική διαδρομή /api/v1/. Οι κύριες συλλογές περιλαμβάνουν /users, /tickets, /categories, /attachments, /auth, /dashboard και /settings. Η πρόσβαση σε μεμονωμένους πόρους γίνεται με την προσθήκη του μοναδικού αναγνωριστικού τους (π.χ. /tickets/{id}), ενώ η πρόσβαση σε εμφωλευμένους πόρους γίνεται λογικά μέσω του γονέα τους (π.χ. /tickets/{id}/comments).

3.3.2. Μεταφορά δεδομένων και Payloads (JSON)

Όλα τα δεδομένα ανταλλάσσονται χρησιμοποιώντας τη μορφή JavaScript Object Notation (JSON). Χρησιμοποιήθηκαν αντικειμένα μεταφοράς δεδομένων (DTO) για όλα τα αιτήματα και τις απαντήσεις του API. Αυτή η προσέγγιση ενισχύει την ασφάλεια, αποτρέποντας ευπάθειες μαζικής ανάθεσης και δημιουργεί μια σαφή, επίσημη σύμβαση για τη δομή των σωμάτων των αιτημάτων, η οποία επιβάλλεται στην επικύρωση από την πλευρά του διακομιστή.

3.3.3. Τυποποιημένη διαχείριση σφαλμάτων

Για να εξασφαλιστεί μια συνεπής και προβλέψιμη εμπειρία για τον χρήστη, εφαρμόστηκε ένας κεντρικός μηχανισμός διαχείρισης εξαιρέσεων/σφαλμάτων χρησιμοποιώντας ένα annotation Spring @ControllerAdvice. Αυτός ο κεντρικός μηχανισμός ονομάστηκε GlobalExceptionHandler, επεμβαίνει στην διαδικασία επιστροφής μηνύματος σε συγκεκριμένες εξαιρέσεις που προκύπτουν από το επίπεδο υπηρεσιών και τις αντιστοιχίζει σε μια τυποποιημένη μορφή απόκρισης που έχει την μορφή JSON και ονομάστηκε ως ApiError.

Το αντικείμενο ApiError παρέχει έναν κωδικό σφάλματος (errorCode) και ένα μήνυμα σφάλματος (errorMessage). Το errorCode προορίζεται για χρήση από το εξωτερικό σύστημα που καταναλώνει το API (στην περίπτωση της εργασίας, το Frontend) ώστε να μπορεί να φιλτράρει τα

σφάλματα και να αναγνωρίζει συγκεκριμένες περιπτώσεις. Το errorMessage αποτελεί το μήνυμα που θα δει ο χρήστης στην οθόνη του. Ένα παράδειγμα απάντησης σε σφάλμα φαίνεται παρακάτω:

```
1. { "errorCode": "ENTITY_NOT_FOUND", "errorMessage": "Ticket not found with ID: 123" }
```

Αυτή η αρχιτεκτονική εγγυάται ότι το frontend θα λαμβάνει πάντα ένα σταθερά δομημένο payload σφάλματος, απλοποιώντας τη λογική χειρισμού σφαλμάτων από την πλευρά του πελάτη.

3.3.4. Προδιαγραφή API endpoint

Η ομοιόμορφη διεπαφή του API ορίζεται από την τυποποιημένη χρήση μεθόδων HTTP, διαδρομών URI και κωδικών κατάστασης. Η λεπτομερής δομή των τελικών σημείων API παρουσιάζεται στον Πίνακας 3.1. Στον πίνακα αναφέρονται οι όροι pagination & filters. Το pagination είναι μια λειτουργία του JPA, συγκεκριμένα του JpaSpecificationExecutor, η οποία επιτρέπει την επιστροφή λίστας από αποτελέσματα στη βάση ορίζοντας το πλήθος των αποτελεσμάτων σε σελίδες. Τα filters είναι τα φίλτρα που χρησιμοποιούνται για να φιλτράρουν τα αποτελέσματα και να επιστραφούν μόνο αυτά που επιθυμεί ο χρήστης.

Πίνακας 3.1 Προδιαγραφή των Βασικών Τελικών Σημείων (Endpoints) του RESTful API

HTTP μέθοδος	**Path**	**Περιγραφή**	**Επίπεδο πρόσβασης (Ρόλοι)**
Αυθεντικοποίηση			
POST	/auth/authenticate	Αυθεντικοποιεί έναν χρήστη και επιστρέφει JWTs.	Δημόσιο
POST	/auth/refresh-token	Δημιουργεί ένα νέο access token από κάποιο ήδη υπάρχουν refresh token.	Αυθεντικοποιημένοι χρήστες
Χρήστες			
GET	/users/getAllUsers	Επιστρέφει μια φιλτραρισμένη λίστα από χρήστες με pagination.	ADMIN
GET	/users/search-customers?q={term}	Ψάχνει για χρήστες με τον ρόλο CUSTOMER.	SUPPORT_AGENT, SUPERVISOR, ADMIN
GET	/users/search-staff?q={term}	Ψάχνει για υπαλλήλους στους χρήστες (που δεν έχουν ρόλο CUSTOMER).	SUPPORT_AGENT, SUPERVISOR, ADMIN
GET	/users/id/{id}	Επιστρέφει τις πληροφορίες ενός χρήστη με βάση το ID του.	Αυθεντικοποιημένοι χρήστες
GET	/users/getAllRoles	Επιστρέφει μια λίστα με όλους τους	Public/Authenticated

		διαθέσιμους ρόλους των χρηστών.	
POST	/users/createUser	Δημιουργεί έναν νέο χρήστη.	ADMIN
POST	/users/changePassword	Επιτρέπει σε έναν αυθεντικοποιημένο χρήστη να αλλάξει τον κωδικό του.	Αυθεντικοποιημένοι χρήστες
PUT	/users/{userId}	Ενημερώνει τα στοιχεία firstName, lastName και role ενός χρήστη	ADMIN
DELETE	/users/{id}	Διαγράφει κάποιον χρήστη από το σύστημα.	ADMIN
Tickets			
GET	/tickets	Επιστρέφει μια φιλτραρισμένη λίστα από tickets με pagination.	Αυθεντικοποιημένοι χρήστες
GET	/tickets/{id}	Επιστρέφει τις πληροφορίες ενός ticket.	Αυθεντικοποιημένοι χρήστες
POST	/tickets	Δημιουργεί ένα νέο ticket (χρήση από Customers)	CUSTOMER
POST	/tickets/on-behalf-of	Δημιουργεί ένα νέο ticket για κάποιον Customer (χρήση από agents)	SUPPORT_AGENT, SUPERVISOR, ADMIN
PATCH	/tickets/{id}	Ενημερώνει τις πληροφορίες ενός ticket.	SUPPORT_AGENT, SUPERVISOR, ADMIN
DELETE	/tickets/{id}	Διαγράφει ένα ticket.	ADMIN
Σχόλια			
GET	/tickets/{ticketId}/comments	Επιστρέφει τα σχόλια για κάποιο ticket.	Αυθεντικοποιημένοι χρήστες
POST	/tickets/{ticketId}/comments	Προσθέτει ένα νέο comment σε κάποιο ticket.	Αυθεντικοποιημένοι χρήστες
Έγγραφα			
POST	/tickets/{ticketId}/attachments	Προσθέτει ένα ή περισσότερα έγγραφα σε κάποιο ticket.	Αυθεντικοποιημένοι χρήστες
POST	/attachments/upload/{ticketId}	Προσθέτει 1 έγγραφο σε κάποιο ticket.	Αυθεντικοποιημένοι χρήστες

POST	/attachments/upload-multiple/{ticketId}	Προσθέτει πολλαπλά έγγραφα σε κάποιο ticket.	Αυθεντικοποιημένοι χρήστες
GET	/attachments/{id}	Κατεβάζει ένα συγκεκριμένο έγγραφο.	Αυθεντικοποιημένοι χρήστες
GET	/attachments/support-request/{ticketId}	Επιστρέφει τις πληροφορίες των εγγράφων σε κάποιο ticket.	Αυθεντικοποιημένοι χρήστες
Κατηγορίες			
GET	/categories	Επιστρέφει λίστα με όλες τις κατηγορίες με pagination.	ADMIN
GET	/categories/active	Επιστρέφει λίστα με τις ενεργές κατηγορίες.	Αυθεντικοποιημένοι χρήστες
POST	/categories	Δημιουργεί μια νέα κατηγορία.	ADMIN
PUT	/categories/{id}	Ενημερώνει το όνομα κάποιας κατηγορίας.	ADMIN
PUT	/categories/{id}/enable	Ενεργοποίηση κάποιας ανενεργής κατηγορίας.	ADMIN
DELETE	/categories/{id}	Απενεργοποιεί κάποια κατηγορία.	ADMIN
Dashboard			
GET	/dashboard/stats	Επιστρέφει τα στατιστικά για το dashboard του χρήστη.	Αυθεντικοποιημένοι χρήστες
GET	/dashboard/actionable-tickets	Επιστρέφει μια λίστα με τα σχετικά tickets για το dashboard του χρήστη.	Αυθεντικοποιημένοι χρήστες
Ρυθμίσεις			
GET	/settings/application	Επιστρέφει τις πληροφορίες του συστήματος (όνομα και logo url).	Δημόσιο
PUT	/settings/application	Ενημερώνει τις πληροφορίες του συστήματος.	ADMIN
GET	/settings/logo	Κατεβάζει το τωρινό logo του συστήματος.	Δημόσιο
POST	/settings/logo-upload	Ενημερώνει το logo του συστήματος.	ADMIN

3.4. Αρχιτεκτονική Ασφαλείας (Role-Based Access Control)

Η αρχιτεκτονική ασφαλείας της εφαρμογής διασφαλίζει την εμπιστευτικότητα, την ακεραιότητα και τη διαθεσιμότητα των δεδομένων. Υιοθετήθηκε μια προσέγγιση, που συνδυάζει έναν ισχυρό μηχανισμό ελέγχου ταυτότητας με ένα μοντέλο εξουσιοδότησης. Ολόκληρο το μοντέλο ασφαλείας βασίζεται στις αρχές του Role-Based Access Control (RBAC), το οποίο παρέχει ένα ευέλικτο και διαχειρίσιμο πλαίσιο εφαρμογής των πολιτικών ασφαλείας.

Η εφαρμογή αξιοποιεί τις ολοκληρωμένες δυνατότητες του Spring Security framework, το οποίο είναι ενσωματωμένο στο Spring Boot backend για την προστασία όλων των τελικών σημείων API. Ένα τυπικό JWT χωρίς κατάσταση παραμένει έγκυρο μέχρι τη λήξη της κρυπτογραφικής του ισχύος, ακόμη και αν ο χρήστης έχει αποσυνδεθεί.

Για να αντιμετωπίσει αυτό το ζήτημα ασφάλειας, η εφαρμογή εφαρμόζει μια υβριδική προσέγγιση. Μετά την επιτυχή σύνδεση, δημιουργείται ένα JWT και τα μεταδεδομένα του αποθηκεύονται σε έναν ειδικό πίνακα jwt_token στη βάση δεδομένων, με την ένδειξη «ενεργό». Σε κάθε εισερχόμενη αίτηση API, το σύστημα εκτελεί μια επικύρωση δύο σταδίων: πρώτα, επικυρώνει κρυπτογραφικά την υπογραφή και την ημερομηνία λήξης του token. Εάν είναι έγκυρο, εκτελείται ένας δεύτερος έλεγχος με κατάσταση έναντι του πίνακα jwt_token για να διασφαλιστεί ότι το token δεν έχει ανακληθεί ρητά. Αυτό το υβριδικό μοντέλο παρέχει την αποτελεσματικότητα της επικύρωσης για τις περισσότερες αιτήσεις, σε συνδυασμό με την κρίσιμη δυνατότητα άμεσης ακύρωσης των token, ενισχύοντας έτσι τη συνολική ασφάλεια της εφαρμογής.

Η διαδικασία με την οποία ένας χρήστης αποδεικνύει την ταυτότητά του και διατηρεί τη συνεδρία του είναι μια κρίσιμη ακολουθία γεγονότων, που απεικονίζεται στο Διάγραμμα 3.3.

Η ροή έχει ως εξής:

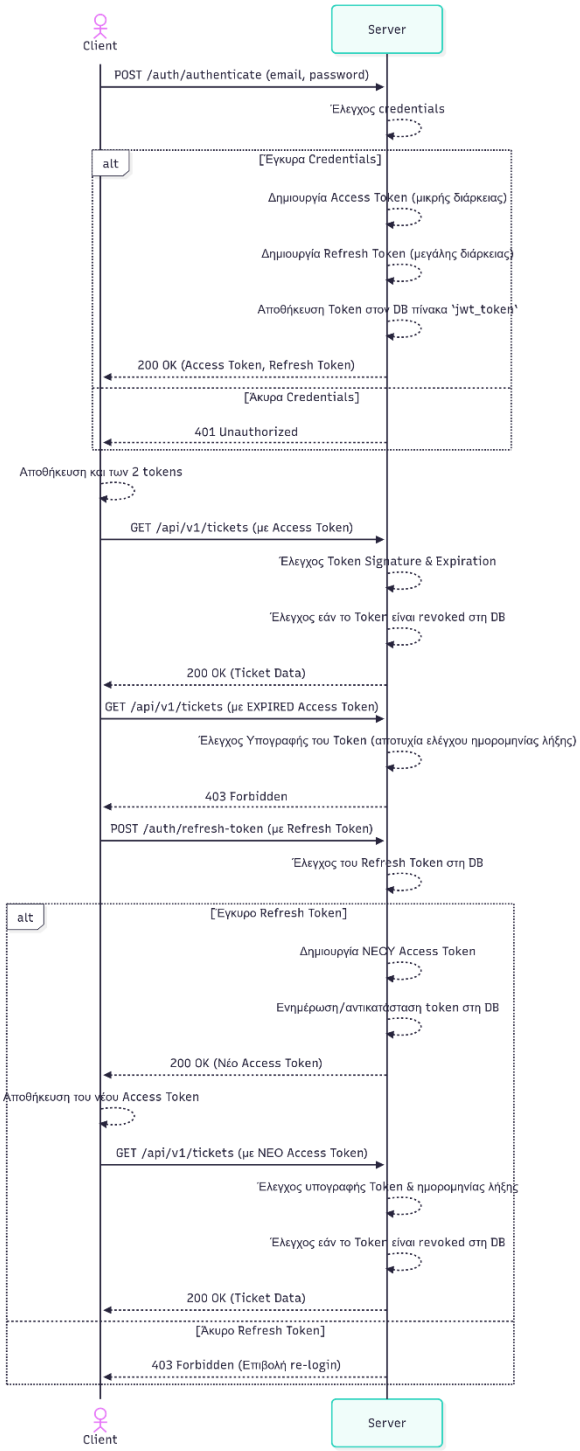
1. Αρχική αυθεντικοποίηση: Ο χρήστης υποβάλλει τα διαπιστευτήριά του (email και κωδικό πρόσβασης) στο τελικό σημείο POST /api/v1/auth/authenticate.
2. Επικύρωση διαπιστευτηρίων: Το backend επικυρώνει αυτά τα διαπιστευτήρια σε σχέση με τον κωδικό πρόσβασης που έχει υποβληθεί σε ασφαλή κρυπτογράφηση στη βάση δεδομένων.
3. Δημιουργία και έκδοση token: Σε περίπτωση επιτυχίας, ο διακομιστής δημιουργεί ένα token πρόσβασης μικρής διάρκειας και ένα token ανανέωσης μακράς διάρκειας. Αυτά αποστέλλονται στον πελάτη.
4. Επόμενες αιτήσεις: Ο πελάτης επισυνάπτει το Access Token στην κεφαλίδα Authorization όλων των επόμενων αιτήσεων στο API.
5. Λήξη και ανανέωση token: Όταν το Access Token λήξει, το API επιστρέφει μια κατάσταση 403 Forbidden. Ο Axios interceptor του πελάτη συλλαμβάνει αυτόματα αυτήν την απόκριση και χρησιμοποιώντας το Refresh Token μακράς διάρκειας υποβάλλει μια σιωπηρή αίτηση στο τελικό σημείο POST /api/v1/auth/refresh-token. Εάν η διαδικασία είναι επιτυχής, ο διακομιστής εκδίδει ένα νέο Access Token και το αρχικό, αποτυχημένο αίτημα API επαναλαμβάνεται αυτόματα, παρέχοντας μια απρόσκοπτη εμπειρία χρήστη.

Η εξουσιοδότηση — η διαδικασία προσδιορισμού των ενεργειών που επιτρέπεται να εκτελεί ένας πιστοποιημένος χρήστης — διαχειρίζεται μέσα από το RBAC. Αυτό το μοντέλο, που τυποποιήθηκε για πρώτη φορά από τους Ferraiolo και Kuhn, αποσυνδέει τα δικαιώματα από τους χρήστες, αναθέτοντάς τα σε αφηρημένους ρόλους.

Το σύστημα ορίζει τέσσερις διακριτούς ρόλους, που υλοποιούνται ως Java Enum: CUSTOMER, SUPPORT_AGENT, SUPERVISOR και ADMIN. Σε κάθε χρήστη εκχωρείται ένας ρόλος, ο οποίος καθορίζει τις δυνατότητές του. Η επιβολή δικαιωμάτων εφαρμόζεται σε δύο επίπεδα για μέγιστη ασφάλεια:

1. Ασφάλεια σε επίπεδο endpoint: Ο έλεγχος πρόσβασης εφαρμόζεται σε επίπεδο controller χρησιμοποιώντας την ασφάλεια σε επίπεδο μεθόδου του Spring Security με το annotation @PreAuthorize. Αυτό παρέχει ένα πρωταρχικό επίπεδο άμυνας (π.χ. διασφαλίζοντας ότι μόνο οι χρήστες με hasRole(“ADMIN”) μπορούν να έχουν πρόσβαση στα endpoints διαχείρισης χρηστών).
2. Ασφάλεια σε επίπεδο service: Λεπτομερής, προγραμματισμένος έλεγχος πρόσβασης εκτελείται και εντός του επιπέδου. Αυτό είναι κρίσιμο για την επαλήθευση της ιδιοκτησίας των δεδομένων. Για παράδειγμα, όταν ένας χρήστης ζητά ένα συγκεκριμένο εισιτήριο, το service layer ελέγχει όχι μόνο τον ρόλο του, αλλά επαληθεύει επίσης ότι το requester_id του εισιτηρίου ταιριάζει με το ID του πιστοποιημένου CUSTOMER. Με αυτόν τον τρόπο ο κάθε χρήστης βλέπει μόνο τις πληροφορίες που πρέπει να δει ανάλογα με τον ρόλο που έχει.

Το πλήρες σύνολο δικαιωμάτων για κάθε ρόλο περιγράφεται λεπτομερώς στον Πίνακας 3.2.



Διάγραμμα 3.3 Διάγραμμα Ακολουθίας για τη Ροή Αυθεντικοποίησης Χρήστη και Ανανέωσης (Token)

Πίνακας 3.2 Πίνακας Δικαιωμάτων βάσει Ρόλων (RBAC Permission Matrix)

Λειτουργία / Ενέργεια	ΠΕΛΑΤΗΣ (CUSTOMER)	ΕΚΠΡΟΣΩΠΟΣ (SUPPORT_AGENT)	ΕΠΟΠΤΗΣ (SUPERVISOR)	ΔΙΑΧΕΙΡΙΣΤΗΣ (ADMIN)
ΑΙΤΗΜΑΤΑ (TICKETS)				
Προβολή Ιδίων Αιτημάτων	☑	Δ/Ε	Δ/Ε	Δ/Ε
Προβολή Όλων των Αιτημάτων	☒	☒	☑	☑
Προβολή Ανατεθειμένων & Μη	Δ/Ε	☑	☑	☑
Δημιουργία Αιτήματος (για εαυτόν)	☑	☒	☒	☒
Δημιουργία Αιτήματος (για πελάτη)	☒	☑	☑	☑
Ενημέρωση Ιδιοτήτων Αιτήματος	☒	☑	☑	☑
Ανάθεση Αιτήματος	☒	☑	☑	☑
Κλείσιμο/Αρχειοθέτηση Αιτήματος	☒	☒	☒	☑
ΣΧΟΛΙΑ (COMMENTS)				
Προβολή Δημόσιων Σχολίων	☑	☑	☑	☑
Προβολή Εσωτερικών Σημειώσεων	☒	☑	☑	☑
Προσθήκη Δημόσιου Σχολίου	☑	☑	☑	☑
Προσθήκη Εσωτερικής Σημείωσης	☒	☑	☑	☑
ΣΥΝΗΜΜΕΝΑ (ATTACHMENTS)				
Προσθήκη σε Ίδιο Αίτημα	☑	Δ/Ε	Δ/Ε	Δ/Ε
Προσθήκη σε Οποιοδήποτε Αίτημα	☒	☑	☑	☑

Λήψη Συνημμένου	☑	☑	☑	☑
ΧΡΗΣΤΕΣ (USERS)				
Προβολή Άλλων Χρηστών	☒	☒	☑	☑
Δημιουργία Χρήστη	☒	☒	☒	☑
Ενημέρωση Χρήστη (Όνομα/Ρόλος)	☒	☒	☒	☑
Διαγραφή Χρήστη	☒	☒	☒	☑
ΚΑΤΗΓΟΡΙΕΣ (CATEGORIES)				
Προβολή Ενεργών Κατηγοριών	☑	☑	☑	☑
Προβολή Όλων (και ανενεργών)	☒	☒	☒	☑
Δημιουργία / Ενημέρωση Κατηγορίας	☒	☒	☒	☑
Απενεργοποίηση / Ενεργοποίηση	☒	☒	☒	☑
ΣΥΣΤΗΜΑ (SYSTEM)				
Προβολή Ιστορικού (Auditing)	☒	☒	☑	☑
Αλλαγή Ρυθμίσεων Εφαρμογής	☒	☒	☒	☑

(Σημείωση: Το "Δ/Ε" σημαίνει "Δεν Εφαρμόζεται", καθώς η ενέργεια δεν αφορά τον συγκεκριμένο ρόλο.)

3.5. Σχεδίαση Συστήματος Καταγραφής (Event-Driven Auditing)

Μια βασική απαίτηση για κάθε σύστημα καταγραφής επιχειρησιακού επιπέδου είναι η δυνατότητα διατήρησης ενός πλήρους και αμετάβλητου ιστορικού όλων των σημαντικών τροποποιήσεων των δεδομένων. Για να ικανοποιήσει αυτή την απαίτηση, η εφαρμογή ενσωματώνει ένα σύστημα

ελέγχου που έχει σχεδιαστεί για να καταγράφει όλες τις αλλαγές κατάστασης στις οντότητες SupportTicket.

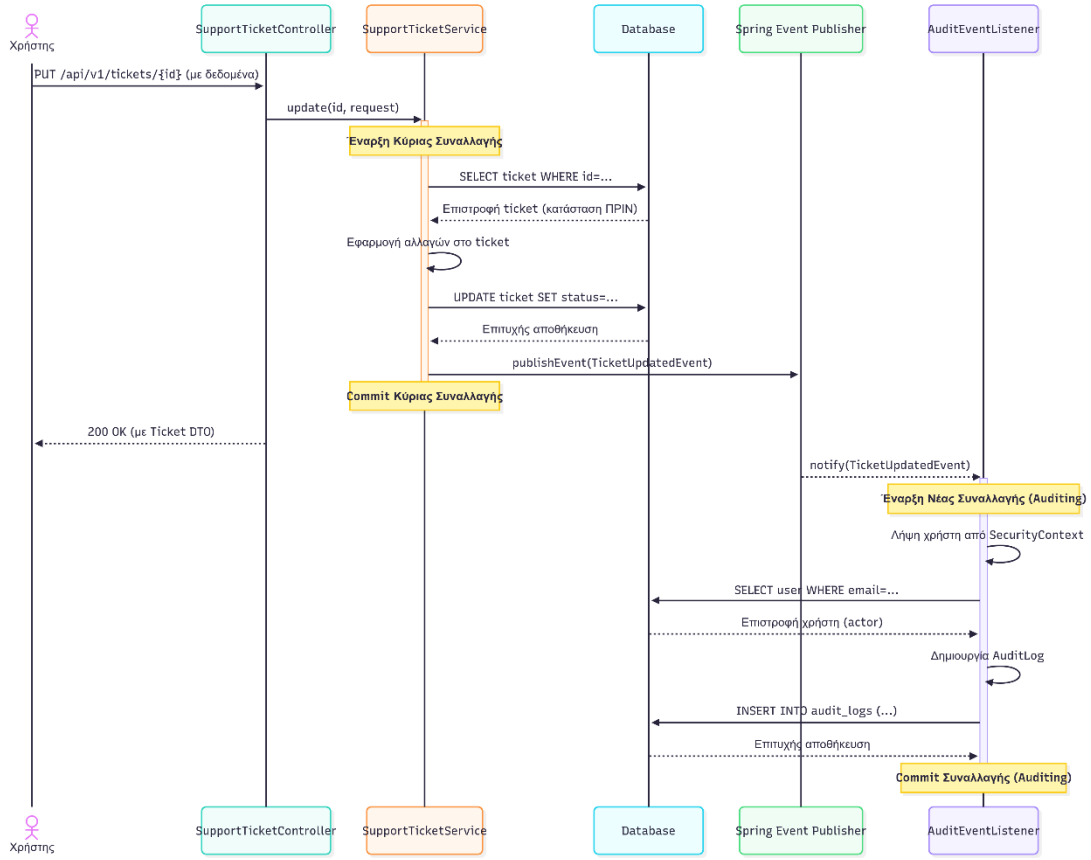
Αρχικά, εξετάστηκε μια κοινή προσέγγιση που χρησιμοποιούσε JPA (Jakarta Persistence API) Entity Listeners με σχολιασμούς όπως @PostUpdate και @PostPersist. Αυτή η προσέγγιση είναι ελκυστικό λόγω της αυτοματοποίησής του. Η λογική ελέγχου ενεργοποιείται αυτόματα από το επίπεδο επιμονής (persistence) κάθε φορά που αποθηκεύεται μια οντότητα. Ωστόσο, η ανάλυση αποκάλυψε σημαντικά προβλήματα ταυτόχρονης πρόσβασης σε αυτήν την προσέγγιση. Η εκτέλεση λειτουργιών βάσης δεδομένων (όπως η αναζήτηση των λεπτομερειών χρήστη του actor ή η αποθήκευση του αρχείου καταγραφής ελέγχου) από ένα callback στον κύκλο ζωής του JPA μπορεί να οδηγήσει σε αδιέξοδα της βάσης δεδομένων (PessimisticLockException) ή σφάλματα κατάστασης συνεδρίας Hibernate (AssertionFailure). Αυτό συμβαίνει επειδή οι λειτουργίες του ακροατή έρχονται σε σύγκρουση με την κύρια συναλλαγή, η οποία βρίσκεται ακόμη σε διαδικασία δέσμευσης.

Για να ξεπεραστούν αυτά τα κρίσιμα ελαττώματα, σχεδιάστηκε και υλοποιήθηκε μια πιο εξελιγμένη, αποσυνδεδεμένη λύση βασισμένη σε συμβάντα (events), αξιοποιώντας τις δυνατότητες του Spring framework. Αυτή η προσέγγιση διασφαλίζει ότι η διαδικασία ελέγχου είναι ασφαλής από πλευράς συναλλαγών και δεν παρεμβαίνει στην βασική επιχειρηματική λογική.

Η λύση αποτελείται από τρία βασικά στοιχεία, όπως απεικονίζεται στο διάγραμμα ακολουθίας του Διάγραμμα 3.4:

1. Συμβάντα εφαρμογής (application events): Απλά, αμετάβλητα αντικείμενα εγγραφής (π.χ. TicketCreatedEvent, TicketUpdatedEvent) ορίζονται για να αντιπροσωπεύουν σημαντικά επιχειρηματικά συμβάντα. Αυτά τα συμβάντα λειτουργούν ως φορείς δεδομένων, διατηρώντας την κατάσταση της οντότητας μετά την αλλαγή.
2. Ο εκδότης συμβάντων (event publisher): Το SupportTicketService, το οποίο περιέχει την βασική επιχειρηματική λογική, είναι πλέον υπεύθυνο και για τη λειτουργία του εκδότη συμβάντων. Αφού μια κύρια επιχειρηματική λειτουργία (όπως η δημιουργία ή η ενημέρωση ενός εισιτηρίου) έχει αποθηκευτεί με επιτυχία στη βάση δεδομένων στο πλαίσιο της δικής της συναλλαγής, η υπηρεσία δημοσιεύει το αντίστοιχο συμβάν χρησιμοποιώντας το ApplicationEventPublisher του Spring.
3. Ο ακροατής συμβάντων (event listener): Ένα ειδικό, διαχειριζόμενο από το Spring στοιχείο, το AuditEventListener, είναι υπεύθυνο για την κατανάλωση αυτών των συμβάντων. Οι μέθοδοι ακροατή φέρουν το annotation @TransactionalEventListener. Δίνει εντολή στο Spring framework να καλέσει τη μέθοδο ακροατή μόνο μετά την επιτυχή ολοκλήρωση της κύριας συναλλαγής που δημοσίευσε το συμβάν. Αυτή η κρίσιμη λειτουργία εξαλείφει εντελώς τον κίνδυνο αδιεξόδων στη βάση δεδομένων και συγκρούσεων κατάστασης συνεδρίας, καθώς η λογική ελέγχου εκτελείται σε δική της ξεχωριστή, επακόλουθη συναλλαγή. Μόλις λάβει ένα συμβάν, ο ακροατής δημιουργεί και αποθηκεύει μια οντότητα AuditLog, δημιουργώντας ένα μόνιμο και αξιόπιστο αρχείο της αλλαγής.

Αυτός ο σχεδιασμός που βασίζεται σε συμβάντα παρέχει έναν σαφή διαχωρισμό των αρμοδιοτήτων. Το SupportTicketService είναι υπεύθυνο μόνο για τη βασική λογική του τομέα του, ενώ το AuditEventListener είναι αποκλειστικά υπεύθυνο για το διατομεακό ζήτημα του ελέγχου. Αυτό έχει ως αποτέλεσμα ένα καθαρότερο, πιο εύκολο στη συντήρηση και ασφαλέστερο από πλευράς συναλλαγών σύστημα.



Διάγραμμα 3.4 Διάγραμμα Ακολουθίας για το Event-Driven Σύστημα Καταγραφής (Auditing)

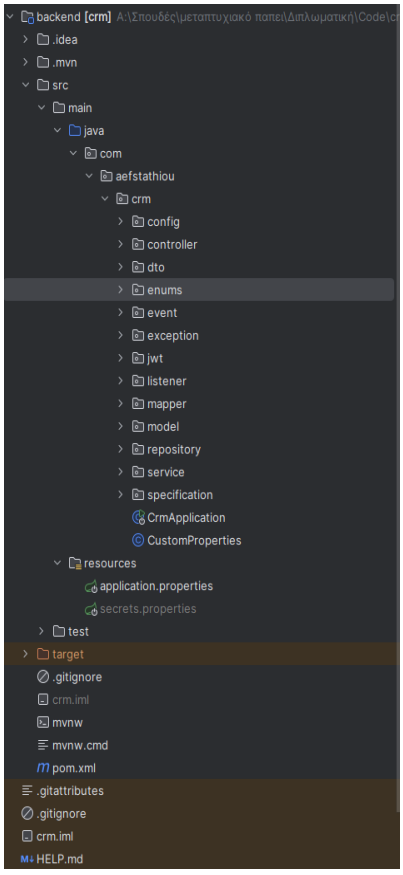
4. Υλοποίηση Συστήματος για τον Διακομιστή (Backend Server)

Αυτό το κεφάλαιο περιγράφει λεπτομερώς την τεχνική υλοποίηση της εφαρμογής από την πλευρά του διακομιστή (backend server) με βάση την αρχιτεκτονική που ορίστηκε στο Κεφάλαιο 3. Το backend αναπτύχθηκε χρησιμοποιώντας το Spring Boot framework (έκδοση 3.5.3) με Java 21, το οποίο επιλέχθηκε για τις δυνατότητες ταχείας ανάπτυξης, το ισχυρό οικοσύστημα και την ισχυρή υποστήριξη για τη δημιουργία ασφαλών RESTful API.

Ο πλήρης πηγαίος κώδικας για το backend, συμπεριλαμβανομένων όλων των οντοτήτων, controller, services και repositories είναι διαθέσιμος στο κοινό για έλεγχο στο Github repository της εργασίας. Ο σύνδεσμος βρίσκεται στο Παράρτημα Α: Repository Πηγαίου Κώδικα.

4.1. Δομή του Project και Βασικές Ρυθμίσεις

Για το backend υιοθετήθηκε μια τυποποιημένη, πολυεπίπεδη αρχιτεκτονική, προκειμένου να εξασφαλιστεί ο σαφής διαχωρισμός των αρμοδιοτήτων, να βελτιωθεί η συντηρησιμότητα και να υπάρχει λογική οργάνωση στον κώδικα. Αυτή η αρχιτεκτονική, η οποία διαχωρίζει την εφαρμογή σε ξεχωριστά επίπεδα για την παρουσίαση (controllers), την επιχειρηματική λογική (services) και την πρόσβαση στα δεδομένα (repositories), αποτελεί θεμελιώδες πρότυπο στο σχεδιασμό επιχειρηματικών εφαρμογών [17].



Εικόνα 4.1 Δομή του Backend Project

Η δομή του project, όπως απεικονίζεται στην Εικόνα 4.1, αντικατοπτρίζει αυτή την πολυεπίπεδη προσέγγιση.

Τα κύρια πακέτα είναι οργανωμένα ως εξής:

- config: Αυτό το πακέτο περιέχει τις κλάσεις Spring @Configuration. Αυτές είναι υπεύθυνες για τον ορισμό των beans και τη ρύθμιση της βασικής υποδομής της εφαρμογής, όπως το SecurityConfiguration που συντονίζει ολόκληρη την αλυσίδα φίλτρων ασφαλείας.
- controller: Αυτό χρησιμεύει ως το επίπεδο παρουσίασης του API. Περιέχει τις κλάσεις @RestController που αντιστοιχίζουν τα εισερχόμενα HTTP request σε συγκεκριμένες μεθόδους από το service. Ο κύριος ρόλος του είναι να χειρίζεται τη σύμβαση του API, να επικυρώνει τα requests και να αναθέτει την επιχειρηματική λογική στο επίπεδο υπηρεσιών.
- dto: Αυτό το πακέτο περιέχει τα Data Transfer Objects, που υλοποιούνται ως Java records. Αυτά τα αντικείμενα ορίζουν τις δημόσιες δομές δεδομένων για αιτήσεις και απαντήσεις API, δημιουργώντας ένα ασφαλές όριο μεταξύ του πελάτη και του μοντέλου.
- enums: Περιέχει τους τύπους Java Enum που χρησιμοποιούνται σε όλη την εφαρμογή, όπως Role, Status και Priority. Αυτοί παρέχουν αναπαραστάσεις σταθερών επιχειρηματικών εννοιών.
- event και listener: Αυτά τα δύο πακέτα αποτελούν τη βάση του συστήματος ελέγχου που βασίζεται σε συμβάντα. Το πακέτο event περιέχει τις αμετάβλητες εγγραφές συμβάντων (π.χ. TicketCreatedEvent), ενώ το πακέτο listener περιέχει το @Component (AuditEventListener) που διαχειρίζεται το Spring και καταναλώνει αυτά τα συμβάντα.
- exception: Περιέχει προσαρμοσμένες κλάσεις εξαιρέσεων (π.χ. ForbiddenException) και το GlobalExceptionHandler (@ControllerAdvice), το οποίο συγκεντρώνει τη διαχείριση σφαλμάτων και διασφαλίζει ότι όλες οι απαντήσεις του API (API responses) έχουν μια συνεπή μορφή σφάλματος.
- jwt: Αυτό το πακέτο είναι αφιερωμένο στην ασφάλεια με JSON Web Token. Περιλαμβάνει το JwtAuthenticationFilter, το οποίο παρεμποδίζει κάθε αίτημα για επικύρωση του token, και το JwtService, το οποίο χειρίζεται τη δημιουργία και την ανάλυση των token.
- mapper: Περιέχει στοιχεία χρησιμότητας που είναι υπεύθυνα για τη λογική μετασχηματισμού μεταξύ αντικειμένων JPA Entity και των αντίστοιχων αναπαραστάσεων DTO. Αυτό επιβάλλει έναν ισχυρό διαχωρισμό μεταξύ του μοντέλου και της σύμβασης του API.
- model: Αυτό είναι το βασικό επίπεδο του μοντέλου. Περιέχει τις κλάσεις JPA @Entity που αντιστοιχούν στους πίνακες της σχεσιακής βάσης δεδομένων, καθορίζοντας τη δομή και τις σχέσεις των δεδομένων της εφαρμογής.
- repository: Αυτό το πακέτο αντιπροσωπεύει το επίπεδο πρόσβασης δεδομένων. Περιέχει τις διεπαφές Spring Data JPA @Repository που αφαιρούν τις αλληλεπιδράσεις της βάσης δεδομένων, παρέχοντας μεθόδους υψηλού επιπέδου για λειτουργίες CRUD.
- service: Αυτό είναι το επίπεδο επιχειρηματικής λογικής. Οι κλάσεις @Service σε αυτό το πακέτο περιέχουν την βασική λογική της εφαρμογής, διαχειρίζονται τις συναλλαγές, επιβάλλουν τους επιχειρηματικούς κανόνες και συντονίζουν τις κλήσεις μεταξύ των βάσεων δεδομένων και άλλων στοιχείων.
- specification: Περιέχει κλάσεις βοηθητικών προγραμμάτων που χρησιμοποιούν το JPA Criteria API για τη δημιουργία δυναμικών, ασφαλών ερωτημάτων βάσης δεδομένων. Αυτό το

μοτίβο χρησιμοποιείται για την υλοποίηση των σύνθετων δυνατοτήτων φιλτραρίσματος στις σελίδες διαχείρισης χρηστών και εισιτηρίων.

Το βασικό configuration της εφαρμογής διαχειρίζεται στον κατάλογο src/main/resources. Ακολουθώντας τις αρχές της Εξωτερικής Διαμόρφωσης, όλες οι ρυθμίσεις που αφορούν το περιβάλλον αφαιρούνται από τον πηγαίο κώδικα. Το αρχείο application.properties περιέχει τυπικές, μη ευαίσθητες παραμέτρους, όπως η διεύθυνση URL της βάσης δεδομένων και η θύρα (port) του διακομιστή. Ένα ξεχωριστό αρχείο secrets.properties, το οποίο αγνοείται από το git, χρησιμοποιείται για την αποθήκευση ευαίσθητων διαπιστευτηρίων, όπως το μυστικό κλειδί JWT (secret key) και ο κωδικός πρόσβασης της βάσης δεδομένων. Αυτή η πρακτική διαχωρισμού της διαμόρφωσης από το εκτελέσιμο αρχείο της εφαρμογής είναι ένα καθιερωμένο πρότυπο για τη δημιουργία φορητών και συντηρήσιμων επιχειρησιακών συστημάτων[4]. Αυτές οι τιμές εισάγονται στην εφαρμογή χρησιμοποιώντας το annotation @Value του Spring, ακολουθώντας την βέλτιστη πρακτική της εξωτερικοποίησης της διαμόρφωσης και της διατήρησης των μυστικών εκτός του version control.

4.2. Υλοποίηση Μοντέλου Δεδομένων (Entities & Repositories)

Το σχήμα της σχεσιακής βάσης δεδομένων, το οποίο σχεδιάστηκε εννοιολογικά στο Κεφάλαιο 3.2, υλοποιήθηκε χρησιμοποιώντας το Java Persistence API (JPA) με το Hibernate ως πάροχο διατήρησης. Αυτή η προσέγγιση επιτρέπει ολόκληρο το μοντέλο της βάσης δεδομένων να οριστεί και να διαχειριστεί μέσω Plain Old Java Objects (POJOs), γνωστών ως οντότητες. Αυτές οι κλάσεις οντοτήτων είναι annotated ώστε να παρέχουν μια αντιστοίχιση μεταξύ του μοντέλου αντικειμένων και των πινάκων της σχεσιακής βάσης δεδομένων.

Το επίπεδο πρόσβασης δεδομένων υλοποιείται χρησιμοποιώντας το Spring Data JPA. Αυτό το framework αφαιρεί περαιτέρω τις λειτουργίες της βάσης δεδομένων, επιτρέποντας τη δημιουργία διεπαφών αποθετηρίου. Με την απλή επέκταση της διεπαφής JpaRepository, η εφαρμογή αποκτά ένα πλήρες σύνολο μεθόδων συναλλαγών Create, Read, Update και Delete (CRUD) χωρίς να απαιτείται η χειροκίνητη υλοποίηση οποιωνδήποτε SQL queries.

Οι ακόλουθες υποενότητες παρουσιάζουν την υλοποίηση των πιο κρίσιμων οντοτήτων και repositories στο σύστημα.

4.2.1. Ανάλυση υλοποίησης για τους Χρήστες (User)

Η οντότητα User έχει σχεδιαστεί για να αντιπροσωπεύει οποιοδήποτε άτομο αλληλεπιδρά με το σύστημα, από εξωτερικούς πελάτες έως εσωτερικούς διαχειριστές. Η υλοποίηση αυτής της οντότητας σχεδιάστηκε με τέτοιο τρόπο ώστε να μπορεί να ενσωματωθεί στο Spring Security framework.

Όπως φαίνεται στον παρακάτω κώδικα, η κλάση User υλοποιεί τη διεπαφή UserDetails που παρέχεται από το Spring Security γεγονός που επιτρέπει στο ίδιο το αντικείμενο User να λειτουργεί ως Principal στο security context μετά από μια επιτυχημένη αυθεντικοποίηση. Αυτό απλοποιεί τη διαδικασία ανάκτησης των στοιχείων του αυθεντικοποιημένου χρήστη σε όλη την εφαρμογή. Τα δικαιώματα του χρήστη ορίζονται από το πεδίο role, το οποίο είναι ένα Enum. Η σημείωση @Enumerated(EnumType.STRING) επιλέχθηκε σκόπιμα για να διατηρηθεί το όνομα του ρόλου (π.χ. «ADMIN») στη βάση δεδομένων, γεγονός που βελτιώνει σημαντικά την αναγνωσιμότητα και τη συντηρησιμότητα των δεδομένων σε σύγκριση με τη χρήση ενός προεπιλεγμένου ακέραιου αριθμού.

Ο κώδικας της οντότητας φαίνεται παρακάτω:

```
@Builder
```

```java
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "users")
@Getter
@Setter
public class User implements UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="user_id")
    private Long id;
    private String firstName;
    private String lastName;
    private String password;
    private String email;
    @Enumerated(EnumType.STRING)
    @Column(name = "role")
    private Role role;
    @OneToMany(mappedBy = "user")
    private List<JwtToken> jwtTokens;

    public User(String firstName, String lastName, String password, String email, Role role) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.password = password;
        this.email = email;
        this.role=role;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return List.of(new SimpleGrantedAuthority("ROLE_" + role.name()));
    }

    @Override
    public String getUsername() {
        return email;
    }

    @Override
    public boolean isAccountNonExpired() {
        return UserDetails.super.isAccountNonExpired();
    }

    @Override
    public boolean isAccountNonLocked() {
        return UserDetails.super.isAccountNonLocked();
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return UserDetails.super.isCredentialsNonExpired();
    }

    @Override
    public boolean isEnabled() {
        return UserDetails.super.isEnabled();
    }

    public String getFullName() {
        return this.firstName + " " + this.lastName;
```

```
    }
}
```

Το επίπεδο πρόσβασης δεδομένων για την οντότητα User διαχειρίζεται το UserRepository. Ως διεπαφή Spring Data JPA, επεκτείνει το JpaRepository για να κληρονομήσει μια πλήρη σειρά τυπικών λειτουργιών CRUD χωρίς να απαιτείται κώδικας υλοποίησης.

Επιπλέον, το repository επεκτείνει το JpaSpecificationExecutor. Αυτό είναι ένα χαρακτηριστικό που επιτρέπει την κατασκευή δυναμικών, ασφαλών database requests. Όπως περιγράφεται λεπτομερώς στην ενότητα 4.3, αυτός ο executor χρησιμοποιείται από το UserService για την κατασκευή της σύνθετης λογικής φιλτραρίσματος πολλαπλών πεδίων που απαιτείται για τη διεπαφή διαχείρισης χρηστών. Το repository περιλαμβάνει επίσης μια custom request μέθοδο, findByEmail, η οποία είναι απαραίτητη για τη διαδικασία ελέγχου ταυτότητας, καθώς επιτρέπει στο UserDetailsService να εντοπίζει αποτελεσματικά έναν χρήστη με βάση το μοναδικό όνομα χρήστη του, το οποίο έχει οριστεί ως το email του.

Ο κώδικας του repository φαίνεται παρακάτω:

```
public interface UserRepository extends JpaRepository<User, Long>,
JpaSpecificationExecutor<User> {

    Optional<User> findByEmail(String email);

}
```

4.2.2. Ανάλυση υλοποίησης για τα Αιτήματα Υποστήριξης (SupportTicket)

Η οντότητα SupportTicket είναι το κεντρικό μοντέλο συναλλαγών στην εφαρμογή, καθορίζοντας τις βασικές σχέσεις που καθοδηγούν τη ροή εργασίας του συστήματος. Η υλοποίησή της ως οντότητα JPA ακολουθεί τα ίδια βασικά πρότυπα με την οντότητα User, αλλά εισάγει μερικά νέα πεδία.

Για να βελτιωθεί η απόδοση της εφαρμογής, χρησιμοποιήθηκε η τεχνική της καθυστερημένης φόρτωσης για τις σχέσεις @ManyToOne με τις οντότητες User και Category, η οποία διαμορφώνεται μέσω της παραμέτρου fetch = FetchType.LAZY. Αυτή η στρατηγική δίνει εντολή στο Hibernate να αναβάλει τη φόρτωση αυτών των σχετικών οντοτήτων έως ότου προσπελαστούν, αποτρέποντας τα προβλήματα αναποτελεσματικών «N+1» queries και διασφαλίζοντας ότι η ανάκτηση λιστών εισιτηρίων παραμένει γρήγορη και αποδοτική από άποψη πόρων.

Επιπλέον, η οντότητα αξιοποιεί το JPA Auditing για να αυτοματοποιήσει τη διαχείριση κρίσιμων χρονικών σημάνσεων του κύκλου ζωής. Τα πεδία createdAt και updatedAt φέρουν τα annotations @CreatedDate και @LastModifiedDate, αντίστοιχα. Με την προσθήκη του AuditingEntityListener σε αυτήν την κλάση, η ευθύνη για τη διατήρηση αυτών των χρονικών σημάνσεων ανατίθεται στο persistence framework. Αυτή είναι μια ανώτερη προσέγγιση σε σχέση με τη χειροκίνητη διαχείριση χρονικών σημάνσεων, καθώς μειώνει τον boilerplate κώδικα στο επίπεδο υπηρεσιών και εγγυάται ότι αυτά τα πεδία είναι πάντα ακριβή και ενημερώνονται με συνέπεια.

Τέλος, η οντότητα αξιοποιεί και τα annotations @PreUpdate & @PrePersist, τα οποία τρέχουν την λογική των μεθόδων που είναι annotated σε αυτές, πριν από την ενημέρωση και δημιουργία της οντότητας αντίστοιχα.

Ο κώδικας της οντότητας φαίνεται παρακάτω:

```
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Entity
```

```java
@EntityListeners({AuditingEntityListener.class})
@Table(name = "support_tickets")
@Getter
@Setter public class SupportTicket {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="support_ticket_id")
    private Long id;
    @Column(nullable = false)
    private String subject;
    @Column(columnDefinition="TEXT")
    private String description;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "requester_id", nullable = false)
    private User requester;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "assigned_to")
    private User assignedTo;
    @Enumerated(EnumType.STRING) private Status status = Status.NEW;
    @Enumerated(EnumType.STRING) private Priority priority = Priority.MEDIUM;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "category_id", nullable = false)
    private Category category;

    @CreatedDate
    @Column(nullable = false, updatable = false)
    private LocalDateTime createdAt;
    @LastModifiedDate
    @Column(nullable = false)
    private LocalDateTime updatedAt;
    private LocalDateTime resolvedAt;

    @OneToMany(mappedBy = "supportTicket", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Attachment> attachments = new ArrayList<>();

    @PreUpdate
    void onUpdate() {
        this.updatedAt = LocalDateTime.now();
    }

    @PrePersist
    void onCreate() {
        this.createdAt = LocalDateTime.now();
        this.updatedAt = this.createdAt;
    }
}
```

Το repository εκτός από την επέκταση του JpaRepository, επεκτείνει επίσης το JpaSpecificationExecutor για να υποστηρίζει τα δυναμικά, βασισμένα σε κριτήρια queries που απαιτούνται από τις δυνατότητες φιλτραρίσματος της εφαρμογής. Το repository περιέχει επίσης αρκετά custom query methods, όπως το findTop10ByAssignedToIsNullAndStatusIsOrderByCreatedAtAsc, τα οποία είναι ειδικά βελτιστοποιημένα για να συμπληρώνουν τα widget του πίνακα ελέγχου με ελάχιστες και αποτελεσματικές κλήσεις στη βάση δεδομένων.

Ο κώδικας του repository φαίνεται παρακάτω:

```
public interface SupportTicketRepository extends JpaRepository<SupportTicket, Long>,
JpaSpecificationExecutor<SupportTicket> {

    long countByRequesterAndStatusNotIn(User user, List<Status> statusList);

    long countByRequesterAndStatus(User user, Status status);

    Long countByAssignedToAndStatusNotIn(User user, List<Status> statusList);

    Long countByAssignedToIsNull();

    long countByStatusNotIn(List<Status> statusList);

    Long countByResolvedAtAfter(LocalDateTime localDateTime);

    List<SupportTicket> findTop5ByRequesterOrderByUpdatedAtDesc(User requester);

    List<SupportTicket> findTop10ByAssignedToAndStatusInOrderByPriorityDescUpdatedAtAsc(User
assignee, List<Status> activeStatuses);

    List<SupportTicket> findTop10ByAssignedToIsNullAndStatusIsOrderByCreatedAtAsc(Status
status);

    List<SupportTicket> findAllByStatusAndResolvedAtBefore(Status status, LocalDateTime
threshold);
}
```

4.2.3. Ανάλυση υλοποίησης για τις κατηγορίες (Category)

Η οντότητα Category παρέχει το δυναμικό σύστημα ταξινόμησης για τα εισιτήρια υποστήριξης. Η υλοποίηση ενός ισχυρού σχήματος κατηγοριοποίησης είναι μια αναγνωρισμένη βασική απαίτηση για εφαρμογές ιστού μεγάλης κλίμακας, καθώς είναι απαραίτητη για τη διευκόλυνση της πλοήγησης των χρηστών και την παροχή των δομημένων δεδομένων που απαιτούνται για πιο προηγμένες τεχνικές εξατομίκευσης [18]. Λήφθηκε μια αρχιτεκτονική απόφαση για την εφαρμογή αυτού του μοντέλου ως δυναμικό JPA @Entity και όχι ως στατικό Enum, παρέχοντας στους διαχειριστές την ευελιξία να διαχειρίζονται τις κατηγορίες μέσω του UI χωρίς να απαιτείται η εγκατάσταση νέου λογισμικού. Η υλοποίησή της ακολουθεί τις ίδιες βασικές αρχές JPA και Spring Data που καθορίστηκαν στις προηγούμενες ενότητες.

Ένα βασικό χαρακτηριστικό αυτής της οντότητας, όπως περιγράφεται στον αρχιτεκτονικό σχεδιασμό στο Κεφάλαιο 3, είναι η παράμετρος boolean isActive για τη διατήρηση της ιστορικής ακεραιότητας των δεδομένων. Όταν ένας διαχειριστής απενεργοποιεί μια κατηγορία, αυτή η παράμετρος ορίζεται ως false, διατηρώντας την εγγραφή της βάσης δεδομένων και διασφαλίζοντας ότι όλες οι υπάρχουσες οντότητες SupportTicket που την αναφέρουν παραμένουν έγκυρες.

Ο κώδικας της οντότητας φαίνεται παρακάτω:

```
@Data
@Entity
@Table(name = "categories")
public class Category {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```

```
    @Column(nullable = false, unique = true)
    private String name;

    @Column(nullable = false)
    private boolean isActive = true;
}
```

Παρόμοια με τα repositories που αναφέρθηκαν προηγουμένως, το CategoryRepository επεκτείνει το JpaRepository. Περιλαμβάνει επίσης μια custom μέθοδο αναζήτησης, findByIsActiveTrue(), η οποία χρησιμοποιείται από το επίπεδο υπηρεσιών για να συμπληρώσει τα αναπτυσσόμενα μενού επιλογής κατηγοριών στη διεπαφή χρήστη, διασφαλίζοντας ότι οι χρήστες μπορούν να αντιστοιχίσουν νέα εισιτήρια μόνο σε κατηγορίες που είναι ενεργές εκείνη τη στιγμή.

Ο κώδικας του repository φαίνεται παρακάτω:

```
public interface CategoryRepository extends JpaRepository<Category, Long>,
JpaSpecificationExecutor<Category> {
    List<Category> findByIsActiveTrue();

    Optional<Object> findByNameIgnoreCase(String name);
}
```

4.2.4. Ανάλυση υλοποίησης για τις υποστηρικτικές οντότητες

Για να ολοκληρωθεί το μοντέλο δεδομένων, υλοποιήθηκαν διάφορες υποστηρικτικές οντότητες για τη διαχείριση της επικοινωνίας, του ελέγχου, της ασφάλειας και της διαμόρφωσης των εφαρμογών. Κάθε οντότητα συνδυάζεται με ένα αντίστοιχο Spring Data JPA repository που παρέχει την απαραίτητη λειτουργικότητα πρόσβασης στα δεδομένα. Αυτά τα στοιχεία ακολουθούν τα ίδια καθιερωμένα πρότυπα JPA και repository που αναφέρθηκαν στις προηγούμενες ενότητες.

- **Οντότητες σχολίων και συνημμένων (Comment & Attachment Entities)**

Η οντότητα Comment παρέχει ένα αμετάβλητο αρχείο για το ιστορικό συνομιλιών ενός εισιτηρίου, με την παράμετρο isInternalNote να διαχωρίζει την δημόσια από την ιδιωτική επικοινωνία. Η οντότητα Attachment αποθηκεύει μεταδεδομένα για τα αρχεία που έχουν μεταφορτωθεί, συνδέοντάς τα με ένα συγκεκριμένο SupportTicket.

Παρακάτω φαίνονται οι κώδικες από τις οντότητες και τα repositories τους:

```
@Data
@Entity
@Table(name = "comments")
public class Comment {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Lob
    @Column(nullable = false, columnDefinition = "TEXT")
    private String content;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "support_ticket_id", nullable = false)
    private SupportTicket supportTicket;
```

```
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "author_id", nullable = false)
    private User author;

    @Column(nullable = false)
    private boolean isInternalNote = false;

    @Column(nullable = false, updatable = false)
    private LocalDateTime createdAt;

    @PrePersist
    protected void onCreate() {
        createdAt = LocalDateTime.now();
    }
}
```

```
public interface CommentRepository extends JpaRepository<Comment, Long> {

    List<Comment> findBySupportTicketIdOrderByCreatedAtAsc(Long ticketId);
}
```

```
@Entity
@Table(name = "attachments")
@Getter
@Setter
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class Attachment {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="attachment_id")
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name="support_ticket_id", nullable=false)
    private SupportTicket supportTicket;

    private String fileName;
    private String mimeType;
    private String filePath;

    private Long sizeBytes;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "uploaded_by", nullable = false)
    private User uploadedBy;
    private LocalDateTime uploadedAt;
}
```

```
public interface AttachmentRepository extends JpaRepository<Attachment, Long> {
    List<Attachment> findBySupportTicket_Id(Long supportRequestId);
}
```

- **Οντότητες AuditLog και JwtToken**

Η οντότητα AuditLog παρέχει ένα αμετάβλητο, βασισμένο σε συμβάντα ιστορικό όλων των αλλαγών κατάστασης ενός εισιτηρίου. Η οντότητα JwtToken χρησιμοποιείται για το υβριδικό μοντέλο

ελέγχου ταυτότητας, αποθηκεύοντας μεταδεδομένα σχετικά με τα εκδοθέντα credentials για να ενεργοποιήσει τη λειτουργία της ανάκλησης credentials για ενισχυμένη ασφάλεια.

Παρακάτω φαίνονται οι κώδικες από τις οντότητες και τα repositories τους:

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "audit_logs")
public class AuditLog {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "support_ticket_id", nullable = false)
    private SupportTicket supportTicket;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id")
    private User actor;

    @Column(nullable = false)
    private String action;

    @Lob
    @Column(columnDefinition = "TEXT")
    private String details;

    @Column(nullable = false, updatable = false)
    private LocalDateTime timestamp;
}
```

```
public interface AuditLogRepository extends JpaRepository<AuditLog, Long> {
    List<AuditLog> findBySupportTicketIdOrderByTimestampDesc(Long ticketId);
}
```

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
public class JwtToken {
    @Id
    @GeneratedValue
    public Integer id;

    @Column(unique = true)
    public String token;

    @Enumerated(EnumType.STRING)
    @Column(name = "token_type", nullable = false, length = 20)
    public TokenType tokenType;

    public boolean revoked;

    public boolean expired;
```

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

```
public interface JwtTokenRepository extends JpaRepository<JwtToken, Integer> {

    @Query("""
      select t from JwtToken t
      where t.user.id = :userId
        and t.tokenType = com.aefstathiou.crm.enums.TokenType.ACCESS
        and t.expired = false
        and t.revoked = false
    """)
    List<JwtToken> findAllValidAccessTokensByUser(Long userId);

    Optional<JwtToken> findByToken(String token);

    @Query("""
      select t from JwtToken t
      where t.token = :token
        and t.tokenType = com.aefstathiou.crm.enums.TokenType.REFRESH
    """)
    Optional<JwtToken> findRefreshToken(String token);
}
```

- **Οντότητα ρυθμίσεων (ApplicationSettings)**

Τέλος, η οντότητα ApplicationSettings είναι ένας πίνακας singleton που έχει σχεδιαστεί για να περιέχει μια μόνο σειρά δεδομένων που μπορούν να διαμορφωθούν σε παγκόσμιο επίπεδο. Αυτό επιτρέπει στους διαχειριστές να διαχειρίζονται δυναμικά την επωνυμία της εφαρμογής, όπως το όνομα και το λογότυπο της εφαρμογής, χωρίς να απαιτούνται αλλαγές στον κώδικα.

Παρακάτω φαίνονται οι κώδικες από την οντότητα και το repository του:

```
@Entity
@Table(name = "application_settings")
@Getter
@Setter
public class ApplicationSettings {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true)
    private Integer singleton = 1;

    @Column(name = "app_name")
    private String appName;

    @Column(name = "logo_url")
    private String logoUrl;

}
```

```
public interface ApplicationSettingsRepository extends JpaRepository<ApplicationSettings,
Long> {

    @Query("SELECT s FROM ApplicationSettings s ORDER BY s.id LIMIT 1")
    Optional<ApplicationSettings> findFirst();
}
```

4.3. Υλοποίηση Controllers και Data Transfer Objects (DTOs)

Το επίπεδο του Controller χρησιμεύει ως το κύριο σημείο εισόδου στο backend της εφαρμογής. Είναι υπεύθυνο για την έκθεση της επιχειρηματικής λογικής που περιέχεται στο επίπεδο των services ως ένα ασφαλές και καλά καθορισμένο σύνολο HTTP endpoints. Η υλοποίηση αυτού του επιπέδου καθοδηγείται από τις αρχές ενός RESTful API, διασφαλίζοντας ότι παραμένει ένα σημείο επαφής μεταξύ των αιτημάτων του πελάτη και της βασικής λειτουργικότητας της εφαρμογής.

Όλοι οι controllers στην εφαρμογή φέρουν το annotation @RestController. Αυτό το Spring annotation είναι μια εξειδίκευση του @Controller που το συνδυάζει με το @ResponseBody και διαμορφώνει αυτόματα όλες τις μεθόδους εντός του controller για να σειριοποιήσει τα αντικείμενα επιστροφής τους απευθείας στο body του HTTP response, συνήθως ως JSON. Αυτό εξαλείφει την ανάγκη για χειροκίνητη μετατροπή της απάντησης σε JSON και απλοποιεί σημαντικά τις μεθόδους του controller. Κάθε controller φέρει επίσης το annotation @RequestMapping σε επίπεδο κλάσης (π.χ. @RequestMapping(«/api/v1/tickets»)) για να ορίσει ένα βασικό URI για όλα τα endpoints που περιέχει.

Ένα κρίσιμο αρχιτεκτονικό πρότυπο είναι η αποκλειστική χρήση αντικειμένων μεταφοράς δεδομένων (DTO)[4] για όλα τα payloads αιτήσεων και απαντήσεων. Οι εσωτερικές οντότητες JPA δεν εκτίθενται ποτέ απευθείας στον τελικό χρήστη. Αυτή είναι μια σκόπιμη επιλογή σχεδιασμού με πολλά πλεονεκτήματα:

1. Ασφάλεια: Χρησιμοποιώντας ειδικά DTO requests όπως SupportTicketCreateRequest και UserUpdateRequest, το API ορίζει μια αυστηρή σύμβαση για τα πεδία που είναι μεταβλητά. Αυτό μειώνει πλήρως τον κίνδυνο ευπαθειών μαζικής εκχώρησης (mass assignment vulnerabilities), μια κατηγορία ευπάθειας που μπορεί να οδηγήσει σε παραβίαση του ελέγχου πρόσβασης (Broken Access Control)[19], εάν δεν αντιμετωπιστεί σωστά.
2. Επικύρωση: Τα DTO αιτήσεων φέρουν annotations με περιορισμούς jakarta.validation (π.χ. @NotBlank, @NotNull). Ο σχολιασμός @Valid στην υπογραφή της μεθόδου του controller ενεργοποιεί το framework επικύρωσης του Spring, διασφαλίζοντας ότι όλα τα εισερχόμενα δεδομένα είναι έγκυρα πριν ακόμη φτάσουν στο επίπεδο υπηρεσιών.
3. Σταθερότητα API: Το επίπεδο DTO αποσυνδέει το δημόσιο συμβόλαιο (contract) API από το εσωτερικό σχήμα της βάσης δεδομένων. Αυτό σημαίνει ότι η εσωτερική οντότητα SupportTicket μπορεί να αναδιαμορφωθεί ή να προστεθούν νέα πεδία χωρίς να παραβιαστεί το υπάρχον contract API για τους πελάτες.

Για endpoints που απαιτούν μεταφόρτωση αρχείων, όπως η δημιουργία εισιτηρίων, οι μέθοδοι του Controller έχουν διαμορφωθεί ειδικά για να χειρίζονται αιτήσεις multipart/form-data. Αυτό επιτυγχάνεται χρησιμοποιώντας το annotation @RequestPart. Ένα μέρος της αίτησης περιέχει το σειριοποιημένο JSON DTO (π.χ. SupportTicketCreateRequest), ενώ τα επόμενα μέρη περιέχουν τα δεδομένα MultipartFile. Αυτό επιτρέπει την αποτελεσματική και τυποποιημένη μεταφορά τόσο δομημένων δεδομένων όσο και περιεχομένου δυαδικών αρχείων σε μία μόνο αίτηση HTTP.

Σε όλες τις μεθόδους του controller που απαιτούν γνώση του τρέχοντα αυθεντικοποιημένου χρήστη, ένα αντικείμενο java.security.Principal εισάγεται ως παράμετρος μεθόδου. Το Spring Security συμπληρώνει αυτόματα αυτό το αντικείμενο με τα στοιχεία του χρήστη που εξάγονται από το επικυρωμένο JWT. Αυτός είναι ο οριστικός και ασφαλής τρόπος για την αναγνώριση του τρέχοντος χρήστη. Στη συνέχεια, το Principal μεταβιβάζεται στο επίπεδο των services, το οποίο το χρησιμοποιεί για την επιβολή επιχειρηματικών κανόνων και ελέγχων ασφαλείας βάσει ιδιοκτησίας. Αυτό αποτρέπει την ανασφαλή πρακτική της εμπιστοσύνης σε ένα αναγνωριστικό χρήστη που αποστέλλεται από τον πελάτη στο σώμα του αιτήματος.

Ο SupportTicketController χρησιμεύει ως η κύρια πύλη για όλες τις λειτουργίες που σχετίζονται με τα αιτήματα υποστήριξης και αποτελεί τον πιο εξελιγμένο controller στην εφαρμογή. Είναι υπεύθυνος για την διαχείριση ολόκληρου του κύκλου ζωής των αιτημάτων, από τη δημιουργία έως τις ενημερώσεις.

Ένα κεντρικό χαρακτηριστικό αυτού του controller είναι ο χειρισμός της λογικής που βασίζεται στους ρόλους μέσω ξεχωριστών, ειδικών endpoints. Όπως φαίνεται στον παρακάτω κώδικα, παρέχονται δύο διακριτές μέθοδοι POST για τη δημιουργία αιτημάτων: μία στη βασική διαδρομή /api/v1/tickets, ασφαλής για τον ρόλο CUSTOMER, και μία άλλη στη διαδρομή /api/v1/tickets/on-behalf-of, ασφαλής για το εσωτερικό προσωπικό. Αυτός ο σχεδιασμός δημιουργεί μια σαφή και ασφαλή σύμβαση (contract) για το API, χρησιμοποιώντας διαφορετικά αντικείμενα μεταφοράς δεδομένων (CustomerTicketCreateRequest και AgentTicketCreateRequest) προσαρμοσμένα σε κάθε περίπτωση χρήσης. Επιπλέον, αυτά τα endpoints δημιουργίας έχουν διαμορφωθεί ώστε να δέχονται αιτήσεις τύπου multipart/form-data χρησιμοποιώντας το annotation @RequestPart, επιτρέποντας την ταυτόχρονη μεταφόρτωση δομημένων δεδομένων του αιτήματος και συνημμένων αρχείων σε ένα ενιαίο, ατομικό αίτημα (atomic request).

Το κύριο GET endpoint για την ανάκτηση της λίστας των αιτημάτων έχει σχεδιαστεί για μέγιστη ευελιξία, καθώς δέχεται ένα αντικείμενο Pageable και πολλαπλές προαιρετικές παραμέτρους @RequestParam. Αυτό επιτρέπει στο frontend να υλοποιήσει ισχυρή σελιδοποίηση, ταξινόμηση και φιλτράρισμα πολλαπλών πεδίων από την πλευρά του εξυπηρετητή (server-side).

Παρακάτω φαίνεται ο κώδικας του SupportTicketController:

```
@CrossOrigin(origins = "http://localhost:5173")
@RestController
@RequestMapping("/api/v1/tickets")
@RequiredArgsConstructor
public class SupportTicketController {

    private final SupportTicketService supportTicketService;

    @PostMapping(consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
    @PreAuthorize("hasRole('CUSTOMER')")
    public SupportTicketDTO createForCustomer(
            @Valid @RequestPart("ticket") CustomerTicketCreateRequest ticket,
            @RequestPart(value = "attachments", required = false) List<MultipartFile>
attachments,
            Principal principal     ) {
        return supportTicketService.createTicketAsCustomer(ticket, attachments, principal);
    }

    @PostMapping(path={"/on-behalf-of"},consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
    @ResponseStatus(HttpStatus.CREATED)
    public SupportTicketDTO createOnBehalfOf(
            @Valid @RequestPart("ticket") AgentTicketCreateRequest ticket,
            @RequestPart(value = "attachments", required = false) List<MultipartFile>
```

```
attachments,
            Principal principal     ) {
        return supportTicketService.createTicketOnBehalfOfCustomer(ticket, attachments,
principal);
    }

    @GetMapping("/{id}")
    public SupportTicketDTO getSupportTicketById(@PathVariable Long id) {
        return supportTicketService.getById(id);
    }

    @GetMapping
    public Page<SupportTicketDTO> getSupportTicketList(
            @RequestParam(required = false) Status status,
            @RequestParam(required = false) Long requesterId,
            @RequestParam(required = false) Long assignedToId,
            @RequestParam(required = false) String subject,
            @RequestParam(required = false) Priority priority,
            Pageable pageable     ) {
        return supportTicketService.list(subject, priority, status, requesterId, assignedToId,
pageable);
    }

    @PatchMapping("/{id}")
    public SupportTicketDTO updateSupportTicket(
            @PathVariable Long id,
            @Valid @RequestBody SupportTicketUpdateRequest update,
            Principal principal     ) {
        return supportTicketService.update(id, update, principal);
    }

    @DeleteMapping("/{id}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    @PreAuthorize("hasRole('ADMIN')")
    public void deleteSupportTicket(@PathVariable Long id) {
        supportTicketService.delete(id);
    }

    @PostMapping(value = "/{id}/attachments", consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
    public List<AttachmentDTO> addAttachments(
            @PathVariable Long id,
            @RequestPart("attachments") List<MultipartFile> attachments,
            Principal principal) {
        return supportTicketService.addAttachments(id, attachments, principal);
    }
}
```

Το CustomerTicketCreateRequest περιέχει μόνο τα πεδία που επιτρέπεται να παρέχει ένας πελάτης. Συγκεκριμένα, παραλείπει το requesterId, καθώς αυτό καθορίζεται με ασφάλεια στο backend από το Principal του χρήστη. Αντίθετα, το AgentTicketCreateRequest περιλαμβάνει το requesterId, καθώς αυτό είναι ένα υποχρεωτικό πεδίο για έναν πράκτορα που δημιουργεί ένα ticket εκ μέρους ενός πελάτη. Αυτή η αυστηρή, συγκεκριμένη για το πλαίσιο μοντελοποίηση είναι απαραίτητη για την πρόληψη τρωτών σημείων ασφαλείας.

Παρακάτω φαίνονται τα DTO για την δημιουργία ticket από πελάτη και από κάποιον agent:

```
public record CustomerTicketCreateRequest(
        @NotBlank String description,
        @NotBlank String subject,
        Priority priority,
```

```
        Long categoryId
) { }
```

```
public record AgentTicketCreateRequest(
        @NotBlank String description,
        @NotBlank String subject,
        Priority priority,
        Long categoryId,
        Long requesterId,
        Long assignedToId
) { }
```

Για την ολοκλήρωση του API, υλοποιήθηκαν αρκετοί άλλοι, πιο απλοί controllers, ακολουθώντας τις ίδιες αρχές RESTful και τα ίδια πρότυπα ασφάλειας που καθιερώθηκαν από τους κύριους controllers. Αυτοί περιλαμβάνουν:

- Το UserController, το οποίο διαχειρίζεται τις διοικητικές ενέργειες για τους χρήστες και παρέχει ασφαλή τελικά σημεία για αναζήτηση.
- Το AuthenticationController, το οποίο χειρίζεται τα δημόσια τελικά σημεία για τη σύνδεση των χρηστών και την ανανέωση των διακριτικών.
- Το CategoryController, το οποίο παρέχει πλήρη λειτουργικότητα CRUD για την οντότητα Category, ασφαλισμένη για τους διαχειριστές.
- Πρόσθετοι ελεγκτές για τη διαχείριση σχολίων, συνημμένων, δεδομένων του πίνακα ελέγχου και ρυθμίσεων εφαρμογής.

Η πλήρης υλοποίηση για όλους τους controllers είναι διαθέσιμη για έλεγχο στο repository πηγαίου κώδικα της διατριβής, όπως περιγράφεται λεπτομερώς στο Παράρτημα Α: Repository Πηγαίου Κώδικα.

4.4. Υλοποίηση Επιπέδου Υπηρεσιών (Service Layer) και Επιχειρησιακής Λογικής

Το Service Layer βρίσκεται μεταξύ του Controller Layer και του Data Access Layer. Είναι υπεύθυνο για την διαχείριση όλης της βασικής επιχειρηματικής λογικής, την οργάνωση των λειτουργιών δεδομένων και τη διαχείριση της ακεραιότητας των συναλλαγών. Όλες οι κλάσεις σε αυτό το επίπεδο είναι σχολιασμένες με το στερεότυπο @Service του Spring, καθιστώντας τις singleton beans που διαχειρίζεται το Inversion of Control (IoC) container της εφαρμογής. Το IoC είναι μια αρχή σχεδιασμού, σύμφωνα με την οποία το πλαίσιο είναι υπεύθυνο για τη δημιουργία και τη σύνδεση των αντικειμένων και των εξαρτήσεών τους, αντί τα αντικείμενα να δημιουργούν τα ίδια τις εξαρτήσεις [20].

Μια βασική ευθύνη αυτού του επιπέδου είναι η διαχείριση των συναλλαγών της βάσης δεδομένων. Όλες οι δημόσιες μέθοδοι που εκτελούν τροποποίηση δεδομένων φέρουν την επισήμανση @Transactional. Αυτή η προσέγγιση αναθέτει την ευθύνη της έναρξης, της δέσμευσης ή της αναίρεσης των συναλλαγών στο Spring Framework, εξασφαλίζοντας τη συνέπεια των δεδομένων και τη συμμόρφωση με τις ιδιότητες ACID (Atomicity, Consistency, Isolation, Durability) που εγγυώνται την αξιοπιστία των δεδομένων σε ένα σύστημα σχεσιακής βάσης δεδομένων [14].

Το SupportTicketService είναι η πιο ολοκληρωμένη υπηρεσία στην εφαρμογή, καθώς υλοποιεί τις πολύπλοκες ροές εργασίας και τις αυτοματοποιημένες μεταβάσεις κατάστασης για ολόκληρο τον κύκλο ζωής των εισιτηρίων. Παρουσιάζει διάφορα πρότυπα υλοποίησης:

- **Ενθυλάκωση επιχειρηματικής λογικής:** Περιέχει τη λογική για τη δημιουργία εισιτηρίων με γνώση ρόλων (διακρίνοντας μεταξύ createTicketAsCustomer και createTicketOnBehalfOfCustomer) και για τη διαχείριση ενημερώσεων.
- **Αυτοματοποίηση ροής εργασίας:** Περιέχει τους κανόνες που αλλάζουν αυτόματα την κατάσταση ενός εισιτηρίου με βάση τις ενέργειες του χρήστη, όπως η μετακίνηση ενός εισιτηρίου από NEW σε OPEN κατά την ανάθεση ή από WAITING_CUSTOMER πίσω σε IN_PROGRESS μετά από απάντηση του πελάτη.
- **Δημοσίευση συμβάντων:** Αυτή η υπηρεσία λειτουργεί ως εκδότης συμβάντων. Μετά την επιτυχή ολοκλήρωση μιας βασικής επιχειρηματικής λειτουργίας (όπως η δημιουργία ή η ενημέρωση ενός εισιτηρίου), δημοσιεύει ένα αντίστοιχο συμβάν (π.χ. TicketCreatedEvent). Αυτό αποσυνδέει τη βασική επιχειρηματική λογική από διατομεακά ζητήματα όπως ο έλεγχος, οδηγώντας σε μια καθαρότερη και πιο εύκολα συντηρήσιμη αρχιτεκτονική.
- **Δυναμική οργάνωση ερωτημάτων:** Χρησιμοποιεί το JPA Specification για να κατασκευάζει και να εκτελεί δυναμικά σύνθετα ερωτήματα αναζήτησης με πολλαπλούς παραμέτρους, όπως φαίνεται στη μέθοδο list.

Ένα βασικό παράδειγμα της ευθύνης της είναι η μέθοδος update, που παρουσιάζεται παρακάτω. Αυτή η μέθοδος ενσωματώνει τη λογική της επιχείρησης για την εφαρμογή αλλαγών από ένα DTO αιτήματος, τη διαχείριση αυτοματοποιημένων μεταβάσεων κατάστασης (όπως η ρύθμιση της χρονικής σήμανσης resolvedAt) και τη σύγκριση των καταστάσεων «πριν» και «μετά» της οντότητας για τη δημιουργία ενός λεπτομερούς μηνύματος ελέγχου. Αφού αποθηκευτεί με επιτυχία η συναλλαγή της βάσης δεδομένων, δημοσιεύει ένα TicketUpdatedEvent, αποσυνδέοντας τη βασική λογική ενημέρωσης από το ζήτημα του ελέγχου.

Παρακάτω φαίνεται το κομμάτι του κώδικα από τον SupportTicketController με τις μεθόδους update και list καθώς και την βοηθητική μέθοδο buildAuditDetails:

```
@Transactional public SupportTicketDTO update(Long id, SupportTicketUpdateRequest req,
Principal principal) {
    SupportTicket ticket = supportTicketRepository.findById(id)
            .orElseThrow(() -> new EntityNotFoundException("Ticket not found with ID: " +
id));

    Map<String, Object> beforeState = new HashMap<>();
    beforeState.put("status", ticket.getStatus());
    beforeState.put("priority", ticket.getPriority());
    beforeState.put("assignee", ticket.getAssignedTo());
    beforeState.put("category", ticket.getCategory());
    beforeState.put("subject", ticket.getSubject());

    if (req.subject() != null) {
        ticket.setSubject(req.subject());
    }
    if (req.priority() != null) {
        ticket.setPriority(req.priority());
    }
    if (req.categoryId() != null) {
        Category category = categoryRepository.findById(req.categoryId())
                .orElseThrow(() -> new EntityNotFoundException("Category not found with ID: "
+ req.categoryId()));
        if (!category.isActive()) {
            throw new IllegalArgumentException("Cannot assign a ticket to an inactive
category.");
        }
        ticket.setCategory(category);
```

```
    }
    if (req.assignedToId() != null) {
        User assignedTo = userRepository.findById(req.assignedToId())
                .orElseThrow(() -> new EntityNotFoundException("Assignee not found with ID: "
+ req.assignedToId()));
        ticket.setAssignedTo(assignedTo);
    }

    if (req.status() != null) {
        Status oldStatus = (Status) beforeState.get("status");
        Status newStatus = req.status();

        ticket.setStatus(newStatus);

        boolean isNowResolved = (newStatus == Status.RESOLVED || newStatus == Status.CLOSED);
        boolean wasNotResolvedBefore = (oldStatus != Status.RESOLVED && oldStatus !=
Status.CLOSED);

        if (isNowResolved && wasNotResolvedBefore) {
            ticket.setResolvedAt(LocalDateTime.now());
        }

        if (!isNowResolved && !wasNotResolvedBefore) {
            ticket.setResolvedAt(null);
        }
    }

    SupportTicket updatedTicket = supportTicketRepository.save(ticket);

    String details = buildAuditDetails(beforeState, updatedTicket);
    if (!details.isEmpty()) {
        eventPublisher.publishEvent(new TicketUpdatedEvent(details, updatedTicket));
    }

    return supportTicketDTOMapper.apply(updatedTicket);
}
```

```
private String buildAuditDetails(Map<String, Object> beforeState, SupportTicket afterState) {
    StringBuilder details = new StringBuilder();

    if (!Objects.equals(beforeState.get("status"), afterState.getStatus())) {
        details.append("Changed status from '").append(beforeState.get("status")).append("' to
'").append(afterState.getStatus()).append("'. ");
    }
    if (!Objects.equals(beforeState.get("priority"), afterState.getPriority())) {
        details.append("Changed priority from
'").append(beforeState.get("priority")).append("' to
'").append(afterState.getPriority()).append("'. ");
    }
    if (!Objects.equals(beforeState.get("assignee"), afterState.getAssignedTo())) {
        String oldAssignee = beforeState.get("assignee") != null ? ((User)
beforeState.get("assignee")).getFullName() : "Unassigned";
        String newAssignee = afterState.getAssignedTo() != null ?
afterState.getAssignedTo().getFullName() : "Unassigned";
        details.append("Changed assignee from '").append(oldAssignee).append("' to
'").append(newAssignee).append("'. ");
    }
    if (!Objects.equals(beforeState.get("category"), afterState.getCategory())) {
        String oldCategory = beforeState.get("category") != null ? ((Category)
beforeState.get("category")).getName() : "None";
        String newCategory = afterState.getCategory() != null ?
afterState.getCategory().getName() : "None";
```

```
        details.append("Changed category from '").append(oldCategory).append("' to
'").append(newCategory).append("'. ");
    }
    if (!Objects.equals(beforeState.get("subject"), afterState.getSubject())) {
        details.append("Changed subject to '").append(afterState.getSubject()).append("'. ");
    }

    return details.toString().trim();
}
```

Για να διατηρηθεί ένας σαφής διαχωρισμός των προβλημάτων, η επιχειρηματική λογική της εφαρμογής κατανέμεται σε διάφορες άλλες εξειδικευμένες κατηγορίες υπηρεσιών. Αυτές οι υπηρεσίες ακολουθούν τα ίδια αρχιτεκτονικά πρότυπα, συμπεριλαμβανομένης της διαχείρισης συναλλαγών και του dependency injection.

- **UserService:** Διαχειρίζεται όλη τη λογική που σχετίζεται με τη δημιουργία χρηστών, τις ενημερώσεις και τις λειτουργίες ασφαλείας, όπως οι αλλαγές κωδικών πρόσβασης. Περιέχει επίσης τη δυναμική λογική αναζήτησης για την εύρεση χρηστών με βάση διάφορα κριτήρια.
- **CategoryService:** Περιέχει τους επιχειρηματικούς κανόνες για τη διαχείριση κατηγοριών, συμπεριλαμβανομένης της εφαρμογής της λογικής «soft delete» και «reactivate».
- **CommentService:** Χειρίζεται τη δημιουργία σχολίων και περιέχει την κρίσιμη λογική ασφαλείας για την αποτροπή της δημιουργίας εσωτερικών σημειώσεων από τους πελάτες και την επιβολή κανόνων ιδιοκτησίας.
- **AttachmentService:** Περιέχει τη λογική χαμηλού επιπέδου για την επικύρωση αρχείων, την αποθήκευση αρχείων στο τοπικό σύστημα αρχείων (στον φάκελο uploads) και την παροχή ενός ασφαλούς τρόπου ανάκτησής τους.
- **DashboardService:** Μια εξειδικευμένη υπηρεσία υπεύθυνη για τη συγκέντρωση δεδομένων από πολλαπλά repositories για τη δημιουργία στατιστικών στοιχείων συγκεκριμένων ρόλων για τα dashboards των χρηστών.
- **ApplicationSettingsService:** Διαχειρίζεται τη λογική της επιχείρησης για την ενημέρωση των καθολικών, μοναδικών ρυθμίσεων της εφαρμογής.
- **AuthenticationService**: Αυτή η υπηρεσία είναι αφιερωμένη στη διαδικασία ελέγχου ταυτότητας. Περιέχει τη λογική για την επικύρωση των διαπιστευτηρίων των χρηστών σε σχέση με τη βάση δεδομένων, τη δημιουργία JWT πρόσβασης και ανανέωσης μετά από επιτυχή σύνδεση και τη διαχείριση του μηχανισμού ανανέωσης token.
- **JwtService**: Πρόκειται για μια υπηρεσία χαμηλού επιπέδου που είναι υπεύθυνη για όλες τις κρυπτογραφικές λειτουργίες που σχετίζονται με τα JSON Web Tokens. Οι αρμοδιότητές της περιλαμβάνουν την εξαγωγή claims από ένα token, την επικύρωση της υπογραφής και της λήξης του και τη δημιουργία νέων token.
- **AuditLogService**: Αυτή η υπηρεσία παρέχει μια απλή διεπαφή για την ανάκτηση ιστορικών δεδομένων ελέγχου. Η κύρια μέθοδος της, getAuditLogsForTicket, καλείται από το AuditLogController για να συμπληρώσει την καρτέλα «Ιστορικό» στη σελίδα λεπτομερειών του εισιτηρίου. (Σημείωση: Η δημιουργία αρχείων καταγραφής ελέγχου γίνεται από το AuditEventListener, το οποίο βασίζεται σε συμβάντα και διατηρεί αυτή τη λογική αποσυνδεδεμένη από τις κύριες υπηρεσίες).
- **TicketCleanupService**: Αυτή η υπηρεσία περιέχει αυτοματοποιημένη επιχειρηματική λογική σε επίπεδο συστήματος. Χρησιμοποιεί το annotation @Scheduled του Spring για να εκτελεί μια καθημερινή εργασία που κλείνει αυτόματα τα εισιτήρια που βρίσκονται σε κατάσταση

RESOLVED για ένα συγκεκριμένο χρονικό διάστημα, διασφαλίζοντας ότι το σύστημα παραμένει τακτοποιημένο και οι ροές εργασίας ολοκληρώνονται.

Η πλήρης εφαρμογή για όλες τις υπηρεσίες είναι διαθέσιμη για έλεγχο στο αποθετήριο πηγαίου κώδικα του έργου, όπως περιγράφεται λεπτομερώς στο Παράρτημα Α.

4.5. Υλοποίηση Ασφάλειας με Spring Security

Η ασφάλεια της εφαρμογής υλοποιήθηκε χρησιμοποιώντας το ολοκληρωμένο και επεκτάσιμο **Spring Security** framework. Η υλοποίηση σχεδιάστηκε ώστε να είναι ισχυρή και stateless, σε συμφωνία με την αρχιτεκτονική RESTful του API. Το μοντέλο ασφάλειας αποτελείται από τρία βασικά στοιχεία: την κεντρική διαμόρφωση ασφάλειας, ένα προσαρμοσμένο φίλτρο ελέγχου ταυτότητας JWT και τα beans που είναι υπεύθυνα για τον έλεγχο ταυτότητας των χρηστών.

Η βασική κλάση της ασφάλειας είναι η SecurityConfiguration, με τα annotations @Configuration και @EnableWebSecurity. Αυτή η κλάση ορίζει τη στάση ασφάλειας της εφαρμογής μέσω ενός bean SecurityFilterChain. Όπως φαίνεται στον παρακάτω κώδικα, αυτή η αλυσίδα διαμορφώνεται:

- **Προστασία CSRF**: Η προστασία Cross-Site Request Forgery (CSRF) είναι απενεργοποιημένη. Το CSRF είναι μια ευπάθεια όπου ένας κακόβουλος ιστότοπος μπορεί να προκαλέσει το πρόγραμμα περιήγησης ενός χρήστη να εκτελέσει μια ανεπιθύμητη ενέργεια σε έναν αξιόπιστο ιστότοπο στον οποίο ο χρήστης είναι επί του παρόντος πιστοποιημένος. Οι παραδοσιακές άμυνες CSRF βασίζονται σε stateful μηχανισμούς, όπως τα tokens συγχρονισμού που αποθηκεύονται στη συνεδρία του χρήστη. Δεδομένου ότι αυτή η εφαρμογή χρησιμοποιεί ένα stateless μοντέλο πιστοποίησης, βασισμένο σε tokens, με ξεχωριστό frontend, δεν είναι ευάλωτη σε παραδοσιακές επιθέσεις CSRF με session-riding. Όπως συνιστάται για stateless API, αυτή η προστασία είναι επομένως απενεργοποιημένη υπέρ άλλων μηχανισμών ασφαλείας [21].
- **Διαχείριση περιόδου λειτουργίας:** Η πολιτική δημιουργίας περιόδου λειτουργίας έχει οριστεί σε STATELESS, ενημερώνοντας ρητά το Spring Security να μην δημιουργεί ή διαχειρίζεται περιόδους λειτουργίας HTTP, καθώς η πιστοποίηση γίνεται ανά αίτημα μέσω του JWT.
- **Εξουσιοδότηση αιτήματος HTTP:** Ένα σύνολο κανόνων αντιστοίχισης ορίζει τα δημόσια endpoints. Είναι σημαντικό να γίνει διάκριση για τη διαδρομή /settings, όπου τα αιτήματα GET είναι δημόσια, αλλά τα αιτήματα PUT και POST περνάνε στον τελικό κανόνα .anyRequest().authenticated(), εξασφαλίζοντας την προστασία τους.
- **Ενσωμάτωση αλυσίδας φίλτρων:** Το προσαρμοσμένο JwtAuthenticationFilter καταχωρείται στην αλυσίδα φίλτρων για να εκτελεστεί πριν από το τυπικό UsernamePasswordAuthenticationFilter.

Παρακάτω φαίνεται ο κώδικας του configuration:

```
@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
@EnableMethodSecurity
public class SecurityConfiguration {
```

```
    private final JwtAuthenticationFilter jwtAuthenticationFilter;
    private final AuthenticationProvider authenticationProvider;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws
Exception{

        httpSecurity
                .cors(httpSecurityCorsConfigurer ->
httpSecurityCorsConfigurer.configurationSource(corsConfigurationSource()))
                .csrf(AbstractHttpConfigurer::disable)
                .authorizeHttpRequests(auth -> auth
.requestMatchers("/api/v1/auth/**").permitAll()
                        .requestMatchers(HttpMethod.GET,
"/api/v1/settings/logo","/api/v1/settings/application").permitAll()
                        .anyRequest()
                        .authenticated()
                )
                .sessionManagement(sess ->
sess.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .authenticationProvider(authenticationProvider)
                .addFilterBefore(jwtAuthenticationFilter,
UsernamePasswordAuthenticationFilter.class)
                .logout(logout ->                        logout
.logoutUrl("/api/v1/auth/logout")
                                .addLogoutHandler(new SecurityContextLogoutHandler())
                                .logoutSuccessHandler((request, response, authentication) ->
SecurityContextHolder.clearContext())
                )
        ;
        return httpSecurity.build();
    }

    @Bean
    public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration corsConfiguration = new CorsConfiguration();
        corsConfiguration.setAllowCredentials(true);
        corsConfiguration.setAllowedOrigins(List.of("http://localhost:5173"));
        corsConfiguration.setAllowedHeaders(Arrays.asList("Origin", "Access-Control-Allow-
Origin", "Content-Type",
                "Accept", "Authorization", "Origin, Accept", "X-Requested-With",
                "Access-Control-Request-Method", "Access-Control-Request-Headers"));
        corsConfiguration.setExposedHeaders(Arrays.asList("Origin", "Content-Type", "Accept",
"Authorization",
                "Access-Control-Allow-Origin", "Access-Control-Allow-Origin", "Access-Control-
Allow-Credentials"));
        corsConfiguration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE",
"OPTIONS", "PATCH"));

        UrlBasedCorsConfigurationSource urlBasedCorsConfigurationSource = new
UrlBasedCorsConfigurationSource();
        urlBasedCorsConfigurationSource.registerCorsConfiguration("/**", corsConfiguration);
        return urlBasedCorsConfigurationSource;
    }

}
```

Το JwtAuthenticationFilter είναι ένα προσαρμοσμένο φίλτρο που επεκτείνει το OncePerRequestFilter, εξασφαλίζοντας την εκτέλεσή του ακριβώς μία φορά για κάθε εισερχόμενο αίτημα. Αυτό το φίλτρο είναι ο φύλακας για όλα τα προστατευμένα API endpoints. Η μέθοδος

doFilterInternal, που εμφανίζεται παρακάτω, περιέχει την κύρια λογική για την πιστοποίηση βάσει token:

1. Ελέγχει το header Authorization του εισερχόμενου αιτήματος για ένα token «Bearer».
2. Εάν υπάρχει token, εξάγεται και μεταβιβάζεται στο JwtService.
3. Το JwtService επικυρώνει την υπογραφή και την ημερομηνία λήξης του token.
4. Εάν το token είναι έγκυρο, εξάγεται το email του χρήστη και καλείται το UserDetailsService για να φορτώσει τα στοιχεία του χρήστη από τη βάση δεδομένων.
5. Τέλος, δημιουργείται ένα UsernamePasswordAuthenticationToken και ορίζεται στο SecurityContextHolder, επικυρώνοντας επίσημα τον χρήστη για τη διάρκεια του αιτήματος.

Εάν κάποιο από αυτά τα βήματα αποτύχει, το πλαίσιο ασφαλείας παραμένει κενό και η πρόσβαση απορρίπτεται.

Για να συνδεθούν οι μηχανισμοί του Spring Security με το συγκεκριμένο μοντέλο χρήστη της εφαρμογής, ορίζονται δύο βασικά bean στο SecurityConfiguration.

- **UserDetailsService:** Ορίζεται ένα bean που παρέχει μια υλοποίηση αυτής της διεπαφής. Η μόνη ευθύνη του είναι να παρέχει μια μέθοδο loadUserByUsername, η οποία απλά καλεί το userRepository.findByEmail(email). Αυτό ενώνει το security framework και την βάση δεδομένων των χρηστών.
- **AuthenticationProvider:** Αυτό το bean συνδέει το UserDetailsService με το PasswordEncoder (BCrypt). Όταν ένας χρήστης προσπαθεί να συνδεθεί μέσω του AuthenticationController, αυτός ο πάροχος χρησιμοποιείται για να ανακτήσει τον χρήστη με βάση το email του και να συγκρίνει με ασφάλεια τον κωδικό πρόσβασης που παρέχεται στο αίτημα με τον κωδικό πρόσβασης που είναι αποθηκευμένος στη βάση δεδομένων.

Παρακάτω φαίνεται ο κώδικας του φίλτρου αυτού:

```
@Component
@RequiredArgsConstructor
public class JwtAuthenticationFilter extends OncePerRequestFilter {
    private final JwtService jwtService;
    private final UserDetailsService userDetailsService;
    private final JwtTokenRepository tokenRepository;

    @Override
    protected void doFilterInternal(@NonNull HttpServletRequest request,@NonNull
HttpServletResponse response,@NonNull FilterChain filterChain) throws ServletException,
IOException {
        final String authHeader=request.getHeader("Authorization");
        final String jwt;
        final String userName;
        if(authHeader==null||!authHeader.startsWith("Bearer ")){
            filterChain.doFilter(request,response);
            return;
        }
        jwt=authHeader.substring(7);
        userName=jwtService.extractUsername(jwt);
        if(userName!=null && SecurityContextHolder.getContext().getAuthentication()==null){
            System.out.println(userName);
            UserDetails userDetails=this.userDetailsService.loadUserByUsername(userName);
```

```
            var isTokenValid = tokenRepository.findByToken(jwt)
                    .map(t -> !t.isExpired() && !t.isRevoked())
                    .orElse(false);
            if(jwtService.isTokenValid(jwt,userDetails) && isTokenValid){
                UsernamePasswordAuthenticationToken authenticationToken=new
UsernamePasswordAuthenticationToken(userDetails,null,userDetails.getAuthorities());
                authenticationToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(authenticationToken);
            }
        }
        filterChain.doFilter(request,response);
    }
}
```

Αυτός ο συνδυασμός configuration, προσαρμοσμένου φίλτρου επεξεργασίας και ειδικών bean αυθεντικοποίησης δημιουργεί μια πλήρη και ασφαλή υλοποίηση του υβριδικού μοντέλου JWT που σχεδιάστηκε στο Κεφάλαιο 3: Αρχιτεκτονική και Σχεδίαση Βάσης Δεδομένων.

5. Υλοποίηση Συστήματος για τον Πελάτη - Διαφυλλιστή (Frontend Client-Browser)

Αυτό το κεφάλαιο περιγράφει λεπτομερώς την τεχνική υλοποίηση του frontend πελάτη, μιας σύγχρονης Single-Page Application (SPA). Ο πρωταρχικός στόχος του frontend είναι να παρέχει ένα responsive, εύχρηστο και role-aware περιβάλλον εργασίας χρήστη που χρησιμοποιεί το RESTful API που αναπτύχθηκε στο backend. Η εφαρμογή δημιουργήθηκε χρησιμοποιώντας τη βιβλιοθήκη React, ένα δημοφιλές και ισχυρό εργαλείο για τη δημιουργία περιβάλλοντος εργασίας χρήστη βασισμένου σε components.

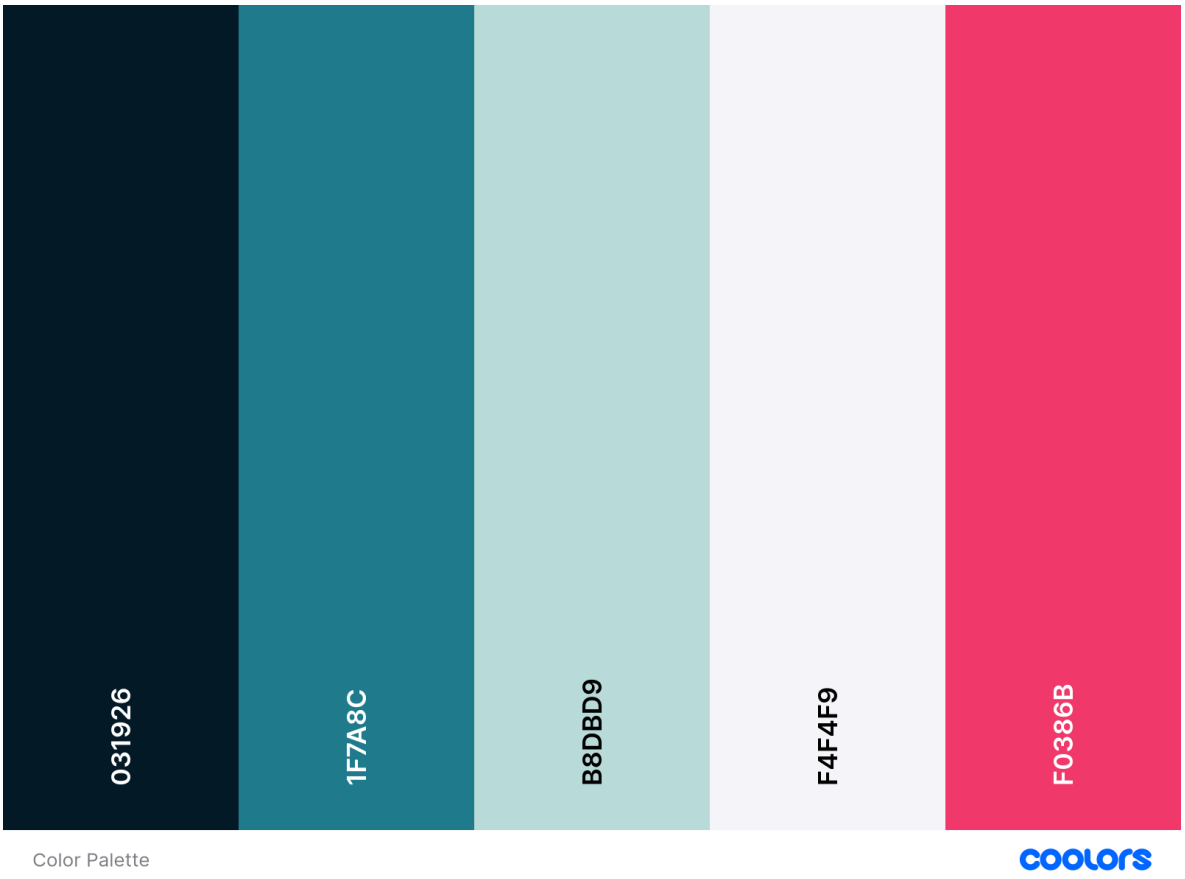
5.1. Χρωματική παλέτα και τυπογραφία

Πριν από την υλοποίηση των επιμέρους στοιχείων, δημιουργήθηκε ένα συνεπές σύστημα σχεδιασμού για να εξασφαλιστεί μια συνεκτική οπτική ταυτότητα για την εφαρμογή. Το σύστημα αυτό αποτελείται από μια καθορισμένη παλέτα χρωμάτων και μια συγκεκριμένη τυπογραφική κλίμακα.

Παλέτα χρωμάτων:

Η παλέτα χρωμάτων επιλέχθηκε ώστε να είναι μοντέρνα, επαγγελματική και προσβάσιμη, με ένα σκούρο θέμα υψηλής αντίθεσης για τα κύρια στοιχεία πλοήγησης και ένα καθαρό, ανοιχτόχρωμο θέμα για τις περιοχές περιεχομένου. Χρησιμοποιήθηκε το site coolors για την γρήγορη παλέτα που προσφέρει [22]. Τα κύρια χρώματα είναι:

- **Navy (#031926):** Χρησιμοποιείται για τα κύρια φόντα, όπως η πλευρική στήλη και η μπάρα πλοήγηση, παρέχοντας μια ισχυρή, σταθερή βάση.
- **Teal (#1F7A8C):** Το κύριο χρώμα έμφασης και αλληλεπίδρασης, που χρησιμοποιείται για ενεργές καταστάσεις, συνδέσμους και κύρια κουμπιά.
- **Aqua (#B8DBD9):** Ένα ελαφρύτερο χρώμα έμφασης που χρησιμοποιείται για καταστάσεις αιωρούμενου (on hover) δείκτη και δευτερεύουσες πληροφορίες.
- **Pink (#F0386B):** Ένα χρώμα έμφασης υψηλής ορατότητας που προορίζεται για κρίσιμες διαδραστικές καταστάσεις, όπως ένα στοιχείο αναπτυσσόμενου μενού στο οποίο έχει τοποθετηθεί ο δείκτης του ποντικιού, για να τραβήξει την προσοχή του χρήστη.
- **Light (#F8F9FA):** Ένα υπόλευκο χρώμα που χρησιμοποιείται για τα φόντα του κύριου περιεχομένου, ώστε να εξασφαλίζεται υψηλή αναγνωσιμότητα.



Εικόνα 5.1 Χρωματική παλέτα εφαρμογής από το site coolors

Τυπογραφία:

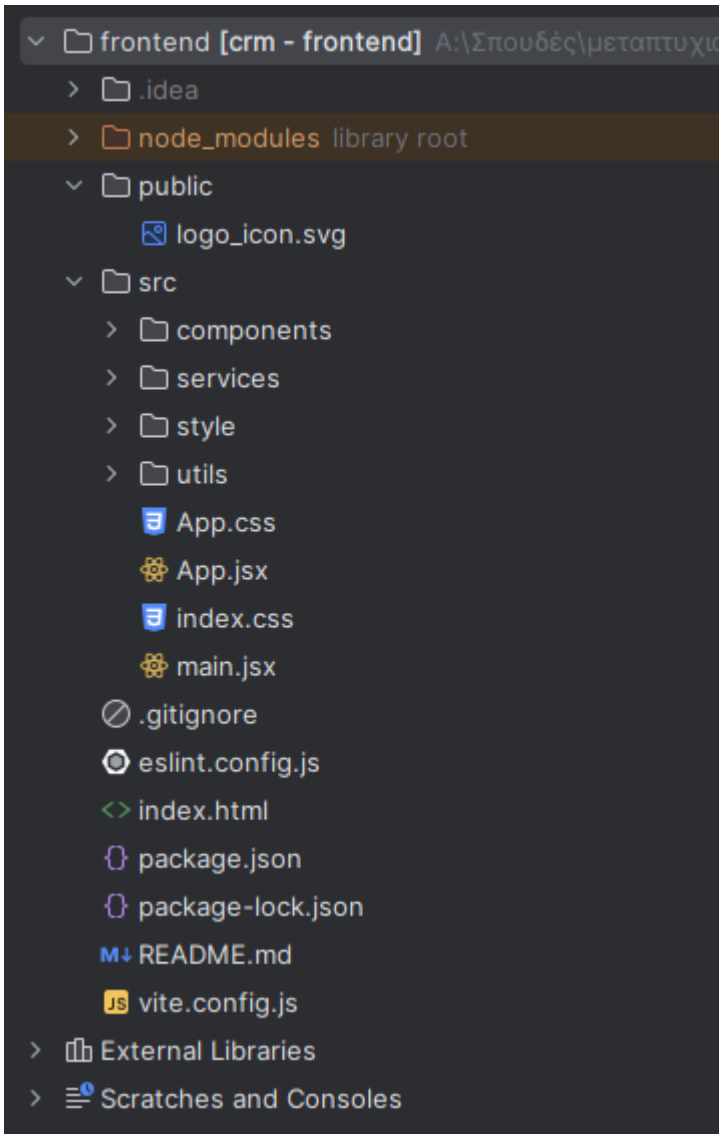
Για να εξασφαλιστεί η μέγιστη συμβατότητα και μια φυσική αίσθηση σε όλα τα λειτουργικά συστήματα, η τυπογραφία της εφαρμογής βασίζεται σε μια σύγχρονη **στοίβα γραμματοσειρών system-ui**. Αυτή η οδηγία CSS δίνει εντολή στον browser να χρησιμοποιήσει την προεπιλεγμένη γραμματοσειρά sans-serif του λειτουργικού συστήματος του χρήστη (π.χ. Segoe UI στα Windows, San Francisco στο macOS). Αυτή η προσέγγιση επιλέχθηκε λόγω της υψηλής απόδοσής της, καθώς δεν απαιτείται η λήψη προσαρμοσμένων γραμματοσειρών, και λόγω της εξαιρετικής αναγνωσιμότητάς της, καθώς χρησιμοποιεί μια γραμματοσειρά που είναι ήδη γνωστή στον χρήστη. Οι επικεφαλίδες αποδίδονται με μεγαλύτερο πάχος (font-weight: 600) για να δημιουργηθεί μια σαφής οπτική ιεραρχία.

5.2. Δομή του Project και Βασικές Βιβλιοθήκες

Το frontend ξεκίνησε με τη χρήση του Vite, ενός σύγχρονου εργαλείου κατασκευής που επιλέχθηκε για τον εξαιρετικά γρήγορο τρόπο κατασκευής των απαραίτητων στοιχείων για να είναι η σελίδα λειτουργική. Υιοθετήθηκε μια τυποποιημένη δομή καταλόγου βασισμένη σε χαρακτηριστικά, προκειμένου να εξασφαλιστεί μια λογική και επεκτάσιμη οργάνωση του κώδικα.

Οι κύριοι φάκελοι εντός του φακέλου src, όπως φαίνεται στην Εικόνα 5.2, είναι οργανωμένοι ως εξής:

- components: Ο φάκελος αυτός περιέχει όλα τα επαναχρησιμοποιήσιμα React components. Τα components οργανώνονται περαιτέρω ανά λειτουργία ή τομέα σε υποκαταλόγους (π.χ. tickets, admin, settings, common, layout) για να βελτιωθεί η δομή και η συντηρησιμότητα.
- services: Αυτός ο φάκελος χρησιμεύει ως επίπεδο πρόσβασης δεδομένων για το frontend. Περιέχει JavaScript modules (π.χ. ticketService.js, userService.js) που ενσωματώνουν όλη τη λογική που βασίζεται σε axios για την πραγματοποίηση κλήσεων API στο backend. Αυτό αφαιρεί τη λογική λήψης δεδομένων από τα στοιχεία UI.
- utils: Περιέχει λειτουργίες χρησιμότητας που χρησιμοποιούνται σε όλη την εφαρμογή, όπως το module roleUtils.js για τη μορφοποίηση και τον έλεγχο των ρόλων των χρηστών.
- style: Αυτός ο φάκελος περιέχει αρχεία CSS που είναι είτε καθολικά (App.css) είτε κοινόχρηστα μεταξύ πολλαπλών στοιχείων (AdminTable.css). Αυτό το project χρησιμοποιεί επίσης μια προσέγγιση για το στυλ, όπου τα στοιχεία έχουν συχνά το δικό τους αποκλειστικό αρχείο CSS εντός του αντίστοιχου υποκαταλόγου στοιχείων (π.χ. TicketDetail.css).
- App.jsx: Το root component της εφαρμογής, υπεύθυνο για τη ρύθμιση του κύριου UserContext.Provider και τον ορισμό της δομής δρομολόγησης σε όλη την εφαρμογή με το React Router.
- main.jsx: Το κύριο σημείο εισόδου της εφαρμογής, όπου το ριζικό στοιχείο React αποδίδεται στο DOM.



Εικόνα 5.2 Δομή του frontend project

Εκτός από το ίδιο το React [12], επιλέχθηκαν αρκετές βασικές βιβλιοθήκες για την επιτάχυνση της ανάπτυξης και την υλοποίηση επαγγελματικών λειτουργιών:

- React Router (react-router-dom): Αυτή είναι η τυπική βιβλιοθήκη για τη διαχείριση της δρομολόγησης από την πλευρά του πελάτη στο React. Επιτρέπει τη δημιουργία ενός πραγματικού SPA, δίνοντας τη δυνατότητα στην εφαρμογή να πλοηγείται μεταξύ διαφορετικών «σελίδων» (components) χωρίς να απαιτείται πλήρης επαναφόρτωση της σελίδας από τον διακομιστή, ενώ διαχειρίζεται χωρίς προβλήματα το ιστορικό του προγράμματος περιήγησης και τις URL παραμέτρους [23].
- React Bootstrap: Αυτή η βιβλιοθήκη παρέχει ένα πλήρες σύνολο προκατασκευασμένων UI components (όπως Button, Modal, Table, Card) που υλοποιούν το σύστημα σχεδιασμού του

Bootstrap. Επιλέχθηκε για την ταχεία δημιουργία ενός καθαρού και πλήρως ανταποκρινόμενου περιβάλλοντος.

- Axios: Ένας ισχυρός, βασισμένος σε υποσχέσεις (promises) HTTP client που χρησιμοποιείται για όλες τις επικοινωνίες με το backend REST API. Η λειτουργία interceptor είναι ένα βασικό χαρακτηριστικό, που επιτρέπει την αυτόματη προσθήκη του JWT Authorization header σε όλες τις εξερχόμενες αιτήσεις και τη διαχείριση της λογικής ανανέωσης token σε παγκόσμιο επίπεδο.
- React Toastify: Χρησιμοποιείται για την εμφάνιση μη παρεμβατικών ειδοποιήσεων «toast» που παρέχουν στους χρήστες feeback για ενέργειες όπως επιτυχημένες ενημερώσεις ή σφάλματα API.
- Recharts: Μια συνθετική βιβλιοθήκη χαρτογράφησης που χρησιμοποιείται για τη δημιουργία οπτικοποιήσεων δεδομένων για τον πίνακα ελέγχου διαχείρισης.

5.3. Διαχείριση Κατάστασης (State Management)

Η αποτελεσματική διαχείριση της κατάστασης είναι ένα πολύ σημαντικό feature μιας ισχυρής εφαρμογής React. Σε μια σύνθετη εφαρμογή όπως αυτή που αναπτύχθηκε στο πλαίσιο της παρούσας διατριβής, η κατάσταση μπορεί να κατηγοριοποιηθεί σε δύο βασικούς τύπους: την **παγκόσμια κατάσταση**, που είναι τα δεδομένα που απαιτούνται σε πολλά στοιχεία (όπως οι πληροφορίες του πιστοποιημένου χρήστη), και την **τοπική κατάσταση**, που είναι τα δεδομένα που αφορούν μόνο ένα στοιχείο (όπως το περιεχόμενο μιας φόρμας εισαγωγής). Αυτή η διατριβή χρησιμοποιεί ένα συνδυασμό ενσωματωμένων εργαλείων του React για την αποτελεσματική διαχείριση και των δύο τύπων κατάστασης.

Για την παγκόσμια κατάσταση, η εφαρμογή αξιοποιεί το ενσωματωμένο **Context API** του React. Αυτό το μοτίβο επιλέχθηκε έναντι εξωτερικών βιβλιοθηκών διαχείρισης κατάστασης (όπως το Redux) λόγω της απλότητάς του και της άμεσης ενσωμάτωσής του στο οικοσύστημα React, το οποίο ταιριάζει απόλυτα στο πεδίο εφαρμογής αυτού του έργου. Το Context API παρέχει έναν τρόπο μεταβίβασης δεδομένων μέσω του component tree χωρίς να χρειάζεται να μεταβιβάζονται χειροκίνητα τα props σε κάθε επίπεδο [24].

Ένα κεντρικό UserContext δημιουργείται στο root της εφαρμογής στο App.jsx. Αυτό το context λειτουργεί ως μοναδική πηγή αλήθειας για όλα τα δεδομένα που έχουν παγκόσμια σημασία, συμπεριλαμβανομένων των εξής:

- isLoggedIn: Μια παράμετρος boolean που υποδεικνύει την κατάσταση αυθεντικοποίησης του χρήστη.
- isLoading: Μια παράμετρος boolean που παρακολουθεί την περίπτωση να τρέχει κάποια διαδικασία που πρέπει να περιμένει το UI, αποτρέποντας τις απότομες ανανεώσεις της σελίδας.
- user: Ένα αντικείμενο που περιέχει τις λεπτομέρειες του συνδεδεμένου χρήστη.
- role: Ο συγκεκριμένος ρόλος του χρήστη, που χρησιμοποιείται για ελέγχους δικαιωμάτων από την πλευρά του πελάτη.
- appName και appLogo: Τα δυναμικά διαμορφωμένα στοιχεία επωνυμίας για την εφαρμογή.

Τα στοιχεία που χρειάζονται πρόσβαση σε αυτήν την παγκόσμια κατάσταση, όπως το TopNav ή το Sidebar, χρησιμοποιούν το hook useContext για να εγγραφούν σε αυτές τις τιμές. Όταν η

παγκόσμια κατάσταση ενημερώνεται (π.χ. μετά την είσοδο ενός χρήστη), όλα τα στοιχεία που την χρησιμοποιούν αναπαράγονται αυτόματα με τα νέα δεδομένα.

Παρακάτω φαίνεται το απόσπασμα κώδικα που περιέχει το UserContext.Provider component:

```
<UserContext.Provider
     value={{
        isLoggedIn,
        setIsLoggedIn,
        user,
        setUser,
        role,
        setRole,
        firstName,
        setFirstName,
        appName,
        setAppName,
        isLoading
     }}
>
…
</UserContext.Provider>
```

Για την τοπική κατάσταση, η εφαρμογή χρησιμοποιεί αποκλειστικά τα ενσωματωμένα **Hooks** του React. Τα δύο κύρια hooks που χρησιμοποιούνται είναι το useState και το useEffect.

- **useState:** Αυτό είναι το βασικό hook για τη διαχείριση της κατάστασης μέσα σε ένα μεμονωμένο στοιχείο. Χρησιμοποιείται εκτενώς σε όλη την εφαρμογή για τη διαχείριση της κατάστασης του UI, όπως η ορατότητα ενός modal (const [showModal, setShowModal] = useState(false);), το περιεχόμενο των εισόδων της φόρμας ή τα δεδομένα που ανακτώνται για μια συγκεκριμένη σελίδα (π.χ. const [tickets, setTickets] = useState([]);).
- **useEffect:** Αυτό το hook χρησιμοποιείται για τη διαχείριση των **παρενεργειών** στα στοιχεία, που είναι λειτουργίες που αλληλεπιδρούν με οτιδήποτε πέρα από την προβολή του UI. Η κύρια περίπτωση χρήσης σε αυτή την διατριβή είναι η λήψη δεδομένων. Στοιχεία όπως το TicketList και το TicketDetail χρησιμοποιούν το useEffect για να ενεργοποιήσουν μια κλήση API στην υπηρεσία του backend όταν το στοιχείο τοποθετείται για πρώτη φορά ή όταν αλλάζει μια εξάρτηση (όπως μια τιμή φίλτρου ή ένας αριθμός σελίδας). Αυτό το hook χρησιμοποιείται επίσης για τη ρύθμιση και τον καθαρισμό των event listeners, όπως το window resize listener στο στοιχείο TopNav.
- **useCallback:** Σε στοιχεία κρίσιμης απόδοσης με πολύπλοκη λογική λήψης δεδομένων (όπως το TicketList), χρησιμοποιείται το hook useCallback. Απομνημονεύει τις λειτουργίες λήψης δεδομένων, διασφαλίζοντας ότι δεν αναδημιουργούνται άσκοπα σε κάθε απόδοση. Αυτό σταθεροποιεί την ταυτότητά τους και αποτρέπει τα υποσυστήματα ή άλλα hooks useEffect από το να προκαλέσουν περιττά re-renders.

Αυτός ο συνδυασμός του Context API για την παγκόσμια κατάσταση και των hooks για την τοπική κατάσταση παρέχει μια ισχυρή, σύγχρονη και εξαιρετικά συντηρήσιμη αρχιτεκτονική διαχείρισης κατάστασης, η οποία είναι τόσο αποτελεσματική όσο και εύκολη στην κατανόηση.

5.4. Υλοποίηση Επαναχρησιμοποιήσιμων Components

Ολόκληρο το frontend της εφαρμογής είναι βασισμένο σε μια **αρχιτεκτονική βασισμένη σε components**, μια βασική αρχή της βιβλιοθήκης React. Αυτή η προσέγγιση περιλαμβάνει τη διάσπαση της διεπαφής χρήστη σε μικρά, ανεξάρτητα και επαναχρησιμοποιήσιμα κομμάτια κώδικα που ονομάζονται components. Αυτή η μεθοδολογία οδηγεί σε μια πιο οργανωμένη, συντηρήσιμη και επεκτάσιμη βάση κώδικα. Αυτή η ενότητα επισημαίνει την υλοποίηση αρκετών βασικών στοιχείων που εξηγούν αυτό το αρχιτεκτονικό πρότυπο.

Για να εξασφαλιστεί μια συνεπής εμφάνιση και αίσθηση σε ολόκληρη την αυθεντικοποιημένη εφαρμογή, δημιουργήθηκε ένα στοιχείο Layout. Αυτό το στοιχείο λειτουργεί ως wrapper για όλες τις ασφαλείς σελίδες και είναι υπεύθυνο για την προβολή του κύριου μέρους της εφαρμογής, το οποίο περιλαμβάνει το TopNav και το Sidebar.

Όπως φαίνεται στον παρακάτω κώδικα, το στοιχείο Layout λαμβάνει άλλα στοιχεία ως υποστοιχεία. Χρησιμοποιεί το UserContext για να προσδιορίσει εάν ένας χρήστης είναι συνδεδεμένος και αποδίδει υπό όρους το Sidebar, διασφαλίζοντας ότι εμφανίζεται μόνο για πιστοποιημένους χρήστες. Αυτό το πρότυπο απομονώνει τη δομή της κύριας διάταξης από το περιεχόμενο της συγκεκριμένης σελίδας, καθιστώντας την εξαιρετικά επαναχρησιμοποιήσιμη και εύκολη στη διαχείριση.

Παρακάτω φαίνεται ο κώδικας του Layout.jsx:

```
import React, {useContext} from 'react';
import Sidebar from './Sidebar.jsx';
import TopNav from './TopNav.jsx';
import '../../style/Layout.css';
import {UserContext} from "./UserContext.jsx";

function Layout({ children, collapsed, setCollapsed, showSidebar }) {
    const { isLoggedIn } = useContext(UserContext);

    return (
        <div className={`layout-container d-flex ${collapsed ? 'collapsed' : ''}`}>
<TopNav/>            <div className="d-flex flex-grow-1 min-h-0" style={{ marginTop: '56px' ,
minHeight: 0  }}>                {showSidebar && isLoggedIn && (
                  <>                          <Sidebar collapsed={collapsed}
setCollapsed={setCollapsed} />                          </>                )}
               <main className="main-content">{children}</main>            </div>
</div>     );
}

export default Layout;
```

Ένα βασικό στοιχείο μιας συντηρήσιμης εφαρμογής είναι η δημιουργία γενικών, επαναχρησιμοποιήσιμων στοιχείων UI. Το DeleteConfirmationModal είναι ένα καλό παράδειγμα ενός τέτοιου στοιχείου. Αντί να δημιουργηθούν ξεχωριστά modal επιβεβαίωσης για τη διαγραφή χρηστών και την απενεργοποίηση κατηγοριών, δημιουργήθηκε ένα ενιαίο, εξαιρετικά διαμορφώσιμο στοιχείο.

Αυτό το στοιχείο έχει σχεδιαστεί ώστε να είναι εντελώς ανεξάρτητο από τα δεδομένα που χειρίζεται. Δέχεται props όπως itemType, itemName, actionText και isPermanent. Αυτά τα props χρησιμοποιούνται για τη δυναμική κατασκευή του τίτλου, του κειμένου του σώματος και των ετικετών των κουμπιών, καθώς και για την υπό όρους απόδοση προειδοποιήσεων (όπως «Αυτή η ενέργεια δεν μπορεί να αναιρεθεί»). Αυτή η προσέγγιση μειώνει σημαντικά την επανάληψη κώδικα και εξασφαλίζει μια συνεπή εμπειρία χρήστη για όλα τα παράθυρα διαλόγου επιβεβαίωσης σε όλη την εφαρμογή.

Παρακάτω φαίνεται ο κώδικας από το DeleteConfirmationModal.jsx:

```
import React from 'react';
import { Modal, Button } from 'react-bootstrap';

const DeleteConfirmationModal = (
    {
        show,
        onHide,
        onConfirm,
        itemType = 'item',
        itemName,
        actionText = 'delete',
        isPermanent = true     }) => {

    const title = `Confirm ${actionText.charAt(0).toUpperCase() + actionText.slice(1)}`;
    const bodyText = `Are you sure you want to ${actionText} the ${itemType}:`;
    const actionButtonText = `Yes, ${actionText.charAt(0).toUpperCase() +
actionText.slice(1)}`;

    return (
        <Modal show={show} onHide={onHide} centered>             <Modal.Header closeButton>
<Modal.Title>{title}</Modal.Title>             </Modal.Header>             <Modal.Body>
<p>{bodyText} <strong>{itemName}</strong> ?</p>                 {isPermanent && (
                    <p className="text-danger">This action cannot be undone.</p>
)}
            </Modal.Body>             <Modal.Footer>                 <Button
variant="secondary" onClick={onHide}>                     Cancel                 </Button>
<Button variant="danger" onClick={onConfirm}>                     {actionButtonText}
                </Button>             </Modal.Footer>         </Modal>     );
};

export default DeleteConfirmationModal;
```

Το στοιχείο TicketList είναι ένα από τα πιο σύνθετα της εφαρμογής, καθώς αποτελείται από πολλά μικρότερα στοιχεία που δημιουργούν ένα περίπλοκο και διαδραστικό πλέγμα δεδομένων. Περιλαμβάνει όλες τις καταστάσεις και τη λογική που απαιτούνται για τη λειτουργικότητά του, όπως:

- Κατάσταση για τα δεδομένα των εισιτηρίων που έχουν ληφθεί, καταστάσεις φόρτωσης και σελιδοποίηση (useState).
- Παρενέργειες για τη λήψη δεδομένων από το API με βάση φίλτρα και ταξινόμηση (useEffect).
- Λειτουργίες λήψης δεδομένων με απομνημόνευση για καλύτερη απόδοση (useCallback).
- Μια ειδική, ανταποκρινόμενη γραμμή φίλτρων που έχει δημιουργηθεί με στοιχεία πλέγματος του react-bootstrap.
- Ένας ανταποκρινόμενος πίνακας που μετατρέπεται σε «προβολή καρτών» σε κινητές συσκευές, προσφέροντας μια ανώτερη εμπειρία χρήστη.
- Η σύνθεση άλλων στοιχείων, όπως το SortableHeader και τα στοιχεία ελέγχου σελιδοποίησης.

Αυτό το στοιχείο αποτελεί ένα ισχυρό παράδειγμα ενός αυτόνομου, πλούσιου σε χαρακτηριστικά «έξυπνου» στοιχείου που είναι κεντρικό για την εμπειρία χρήστη της εφαρμογής. Παρακάτω φαίνεται απόσπασμα από τον κώδικα του αρχείου TicketList.jsx:

```
const DEBOUNCE_DELAY = 500;

function TicketList() {
```

```
    const { user, role } = useContext(UserContext);
    const [tickets, setTickets] = useState([]);
    const [isLoading, setIsLoading] = useState(true);
    const [totalPages, setTotalPages] = useState(0);
    const [currentPage, setCurrentPage] = useState(0);
    const [pageSize, setPageSize] = useState(10);

    const unassignedOption = { value: -1, label: 'Unassigned' };

    const [filters, setFilters] = useState({
        subject: '',
        status: null,
        priority: null,
        requesterId: null,
        assignedToId: null,
        assigneeFilter: 'mine',
    });

    const [sortField, setSortField] = useState('updatedAt');
    const [sortDir, setSortDir] = useState('desc');

    const isCustomer = checkIfHasRole(role, 'ROLE_CUSTOMER');
    const isAgent = checkIfHasRole(role, 'ROLE_SUPPORT_AGENT');
    const isSupervisorOrAdmin = checkIfHasRole(role, ['ROLE_SUPERVISOR', 'ROLE_ADMIN']);

    const fetchTickets = useCallback(async (page, size, currentFilters, field, dir) => {
        setIsLoading(true);
        try {
            const apiFilters = {
                subject: currentFilters.subject,
                status: currentFilters.status,
                priority: currentFilters.priority,
                requesterId: currentFilters.requesterId,
                assignedToId: currentFilters.assignedToId             };

            if (isCustomer) {
                apiFilters.requesterId = user.id;
            } else if (isAgent) {
                if (currentFilters.assigneeFilter === 'mine') {
                    apiFilters.assignedToId = user.id;
                } else if (currentFilters.assigneeFilter === 'unassigned') {
                    apiFilters.assignedToId = -1;
                }
            }

            const response = await ticketService.getTickets(page, size, apiFilters, field,
dir);
            setTickets(response.data.content);
            setTotalPages(response.data.page?.totalPages || 0);
        } catch (err) {
            toast.error(err.response.data?.errorMessage || "Failed to fetch tickets.");
        } finally {
            setIsLoading(false);
        }
    }, [user?.id, isCustomer, isAgent]);

    useEffect(() => {
        if (!user) return;
        const handler = setTimeout(() => {
            setCurrentPage(0);
            fetchTickets(0, pageSize, filters, sortField, sortDir);
        }, DEBOUNCE_DELAY);
```

```
        return () => clearTimeout(handler);
    }, [user, filters, pageSize, sortField, sortDir, fetchTickets]);

<Table striped bordered hover responsive className="responsive-table">
     <thead>
          <tr>
               <SortableHeader field="id">ID</SortableHeader>
               <SortableHeader field="subject">Subject</SortableHeader>
               {/* ... κτλ. ... */}
          </tr>
      </thead>
      <tbody>
          {/* ... */}
      </tbody>
</Table>
```

5.5. Επικοινωνία με το API

Όλη η επικοινωνία μεταξύ του React frontend και του Spring Boot backend γίνεται μέσω ασύγχρονων αιτήσεων HTTP στο RESTful API. Για να διαχειριστεί αυτή η επικοινωνία με καθαρό, κεντρικό και εύκολο στη συντήρηση τρόπο, επιλέχθηκε η βιβλιοθήκη **Axios** ως ο κύριος HTTP client.

Αντί να γίνονται άμεσες κλήσεις axios.get() ή axios.post() από τα στοιχεία του UI, υλοποιήθηκε ένα ειδικό **επίπεδο υπηρεσιών** στο frontend. Αυτή η αρχιτεκτονική αφαιρεί όλη τη λογική λήψης δεδομένων από τα components. Κάθε μονάδα υπηρεσιών (π.χ. ticketService.js, userService.js) είναι υπεύθυνη για έναν συγκεκριμένο τομέα του API, παρέχοντας ένα σαφές και επαναχρησιμοποιήσιμο σύνολο λειτουργιών για το υπόλοιπο της εφαρμογής.

Ένα ενιαίο, κεντρικό Axios instance έχει διαμορφωθεί και χρησιμοποιείται σε όλη την εφαρμογή. Αυτό το intance, που ορίζεται σε ένα ειδικό module api.js, ορίζει το baseURL για όλα τα αιτήματα API. Αυτό απλοποιεί τις λειτουργίες της υπηρεσίας, καθώς χρειάζεται μόνο να παρέχουν τη σχετική διαδρομή προς ένα endpoint. Ορίζει επίσης το προεπιλεγμένο Content-Type Header σε application/json, εξασφαλίζοντας συνέπεια για όλα τα εξερχόμενα αιτήματα. Παρακάτω φαίνεται το απόσπασμα του κώδικα που φτιάχνει το instance:

```
const instance = axios.create({
    baseURL: "http://localhost:8080/api/",
    headers: {
        "Content-Type": "application/json",
    },
});
```

Ένα βασικό χαρακτηριστικό του επιπέδου επικοινωνίας API είναι η χρήση ενός **Axios request interceptor**. Ένας interceptor είναι μια λειτουργία που εκτελεί το Axios *πριν* αποσταλεί ένα αίτημα. Ένας interceptor αιτημάτων ρυθμίστηκε ώστε να επισυνάπτει αυτόματα το JWT Access Token στο Authorization Header κάθε εξερχόμενης κλήσης API σε ένα προστατευμένο τελικό σημείο.

Αυτό προσφέρει πολλά οφέλη:

- **Αποσύνδεση:** Τα στοιχεία UI και οι λειτουργίες υπηρεσιών δεν χρειάζεται να γνωρίζουν το token του χρήστη. Απλώς πραγματοποιούν μια κλήση API και ο interceptor χειρίζεται τη λογική αυθεντικοποίησης με διαφάνεια.
- **Ασφάλεια:** Εξασφαλίζει ότι το token ελέγχου ταυτότητας δεν ξεχνιέται ποτέ και εφαρμόζεται με συνέπεια σε όλα τα προστατευμένα αιτήματα.

- **Κεντρική λογική:** Όλη η λογική που σχετίζεται με την προσθήκη της κεφαλίδας Authorization βρίσκεται σε ένα μόνο, προβλέψιμο σημείο.

Παρακάτω φαίνεται το απόσπασμα κώδικα για το συγκεκριμένο interceptor του request:

```
instance.interceptors.request.use(
    (config) => {
        const token = TokenService.getLocalAccessToken();

        const publicUrls = [
            "/v1/auth/authenticate",
            "/v1/auth/refresh-token"         ];

        if (config.url === "/v1/settings/application" && config.method.toLowerCase() ===
'get') {
            return config;
        }

        if (token && !publicUrls.includes(config.url)) {
            config.headers["Authorization"] = 'Bearer ' + token;
        }
        return config;
    },
    (error) => {
        return Promise.reject(error);
    }
);
```

Για να παρέχεται μια σωστή εμπειρία χρήστη, υλοποιήθηκε ένας **Axios response interceptor** για την αυτόματη διαχείριση της λήξης του Access Token. Αυτός ο interceptor ελέγχει κάθε απάντηση που επιστρέφεται από τον διακομιστή.

Εάν ο interceptor εντοπίσει ένα σφάλμα 403 Forbidden (που υποδηλώνει λήξη του Access Token), εκτελεί αυτόματα και αθόρυβα τις ακόλουθες ενέργειες:

1. Παύει το αρχικό, αποτυχημένο αίτημα.
2. Δημιουργεί ένα ξεχωριστό αίτημα POST στο τελικό σημείο /api/v1/auth/refresh-token, χρησιμοποιώντας το μακράς διάρκειας Refresh Token.
3. Εάν η ανανέωση είναι επιτυχής, ενημερώνει τα τοπικά αποθηκευμένα tokens με το νέο Access Token.
4. Τέλος, επαναλαμβάνει αυτόματα το αρχικό, αποτυχημένο αίτημα με το νέο token.

Ολόκληρη αυτή η διαδικασία είναι εντελώς διαφανής για τον χρήστη. Από τη δική τους πλευρά, η συνεδρία δεν διακόπτεται ποτέ, γεγονός που παρέχει μια ομαλή και επαγγελματική εμπειρία χρήστη. Αυτό απλοποιεί επίσης δραματικά τη λογική στα στοιχεία της διεπαφής χρήστη, καθώς δεν χρειάζεται να περιέχουν ατομική διαχείριση σφαλμάτων για τη λήξη του token. Παρακάτω φαίνεται το απόσπασμα κώδικα για το συγκεκριμένο interceptor του response:

```
instance.interceptors.response.use(
    (res) => {
        return res;
    },
    async (err) => {
        const originalConfig = err.config;

        if (originalConfig.url !== "/v1/auth/authenticate" && err.response) {
            if (err.response.status === 403 && !originalConfig._retry) {
```

```
                originalConfig._retry = true;

                const refreshToken = TokenService.getLocalRefreshToken();

                if (refreshToken) {
                    try {
                        const rs = await instance.post("/v1/auth/refresh-token",{}, {
                            headers: {
                                'Authorization': 'Bearer '+TokenService.getLocalRefreshToken()
                            }
                        });

                        TokenService.setTokens(rs.data);
                        return instance(originalConfig);
                    } catch (_error) {
                        console.error("Refresh token is invalid. Forcing logout.", _error);
                        await authService.logout();
                        return Promise.reject(_error);
                    }
                }
                else {
                    console.log("No refresh token available. Logging out.");
                    TokenService.removeTokens();
                }
            }
        }

        return Promise.reject(err);
    }
);

export default instance;
```

5.6. Δρομολόγηση (Routing)

Η πλοήγηση και η διαχείριση προβολών της εφαρμογής πραγματοποιούνται από τη βιβλιοθήκη **React Router** (react-router-dom). Αυτή η βιβλιοθήκη είναι το de facto πρότυπο για την υλοποίηση **δρομολόγησης από την πλευρά του πελάτη** σε εφαρμογές React όπως αναφέρθηκε σε προηγούμενο κεφάλαιο.

Η κύρια διαμόρφωση δρομολόγησης ορίζεται κεντρικά στο στοιχείο App.jsx, όπου κάθε στοιχείο <Route> συνδέει μια διαδρομή, όπως /tickets/:ticketId, με το αντίστοιχο component, όπως το TicketDetail.

Ένας κρίσιμος παράγοντας για την ασφάλεια του frontend της εφαρμογής είναι να διασφαλίζει ότι οι χρήστες δεν μπορούν να έχουν πρόσβαση σε διαδρομές για τις οποίες δεν έχουν εξουσιοδότηση. Αντί να διασκορπίζεται αυτή η λογική σε διαφορετικά στοιχεία, υλοποιήθηκε ένα ειδικό component με το όνομα **ProtectedRoute** για να συγκεντρωθεί και να επιβληθεί ο έλεγχος πρόσβασης σε επίπεδο δρομολόγησης.

Αυτό το στοιχείο Protected λειτουργεί ως πύλη ασφαλείας, περιβάλλει κάθε διαδρομή που απαιτεί αυθεντικοποίηση ή συγκεκριμένους ρόλους. Όπως φαίνεται στον παρακάτω κώδικα, ενσωματώνει διάφορους βασικούς ελέγχους ασφαλείας:

1. **Έλεγχος πιστοποίησης:** Επαληθεύει ότι ο χρήστης είναι συνδεδεμένος ελέγχοντας την ένδειξη isLoggedIn από το global UserContext.

2. **Έλεγχος εξουσιοδότησης:** Δέχεται ένα requiredRoles prop (μια σειρά από συμβολοσειρές ρόλων) και το συγκρίνει με τους ρόλους του τρέχοντος χρήστη για να καθορίσει εάν διαθέτει τα απαραίτητα δικαιώματα.
3. **Διαχείριση κατάστασης φόρτωσης:** Χειρίζεται την αρχική κατάσταση isLoading από το περιβάλλον, εμφανίζοντας έναν δείκτη φόρτωσης και αποτρέποντας πρόωρες ανακατευθύνσεις πριν από την οριστική καθιέρωση της κατάστασης ελέγχου ταυτότητας του χρήστη.
4. **Ανακατεύθυνση και ανατροφοδότηση χρήστη:** Εάν ένας χρήστης δεν έχει πιστοποιηθεί ή δεν διαθέτει τα απαιτούμενα δικαιώματα, το στοιχείο τον ανακατευθύνει αυτόματα σε μια ασφαλή διαδρομή (π.χ. τη σελίδα σύνδεσης) χρησιμοποιώντας το στοιχείο <Navigate> του React Router. Είναι σημαντικό ότι παρέχει επίσης σαφή ανατροφοδότηση χρήστη, αποστέλλοντας μια ειδοποίηση «άδεια απορρίφθηκε», διασφαλίζοντας ότι ο χρήστης δεν θα μείνει ποτέ σε μια κενή ή κατεστραμμένη σελίδα.

Παρακάτω φαίνεται ο κώδικας του ProtectedRoute.jsx:

```
const Protected = ({ requiredRoles = [], isLoading, isLoggedIn, children, redirectTo =
"/login" }) => {
    const {user} = useContext(UserContext);
    const roles = user?.authorities ?? [];

    const hasPermission = requiredRoles.length === 0         ? true         : roles.some(r =>
requiredRoles.includes(r));

    const toastShown = useRef(false);
    const navigate = useNavigate();

    useEffect(() => {
        if (isLoading) return;
        if (!isLoggedIn || !hasPermission) {
            if (!toastShown.current) {
                toast.error("You do not have permission to access this page.");
                toastShown.current = true;
            }
        } else {
            // reset so future denials can show again (optional)
            toastShown.current = false;
        }

    }, [isLoggedIn, hasPermission,isLoading,toastShown]);

    useEffect(() => {
        const onPopState = () => {
            if (!isLoggedIn || !hasPermission) {
                if (!toastShown.current) {
                    toast.error("You do not have permission to access this page.");
                    toastShown.current = true;
                }
                navigate(redirectTo, { replace: true });
            }
        };
        window.addEventListener("popstate", onPopState);
        return () => window.removeEventListener("popstate", onPopState);
    }, [isLoggedIn, hasPermission, navigate, redirectTo]);

    if (isLoading) {
        return (
            <div className="container py-5 text-center">                <div
```

```
className="spinner-border" role="status" />                <div className="mt-
2">Loading…</div>            </div>         );
    }

    if(!user || !isLoggedIn || !hasPermission) {
        return <Navigate to={redirectTo} replace />;
    }

    return children;
};

export default Protected;
```

Αυτή η προσέγγιση προστασίας διαδρομών με βάση τα στοιχεία είναι ένα καθαρό πρότυπο. Όπως φαίνεται στο παράδειγμα χρήσης από το App.jsx παρακάτω, καθιστά την κύρια διαμόρφωση διαδρομών εύκολη στην ανάγνωση και διασφαλίζει ότι οι κανόνες ασφαλείας εφαρμόζονται με συνέπεια σε όλες τις προστατευμένες περιοχές της εφαρμογής.

Παρακάτω φαίνεται απόσπασμα κώδικα από το App.jsx που φαίνονται 4 δημόσια routes και 1 protected:

```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/login" element={<Login />} />
  <Route path="/help" element={<HelpPage />} />
  <Route path="*" element={<PageNotFound />} />

    <Route
        path="/admin/users"
        element={
            <ProtectedRoute
                isLoading={isLoading}
                requiredRoles={["ROLE_ADMIN"]}
                isLoggedIn={isLoggedIn}
             >
                <UserList />
             </ProtectedRoute>
        }
    />
```

6. Δοκιμές και αξιολόγηση λειτουργικότητας

Η μεθοδολογία δοκιμών για την εφαρμογή σχεδιάστηκε ώστε να είναι πρακτική και διεξοδική, διασφαλίζοντας την ορθότητα και την ασφάλεια όλων των λειτουργιών που υλοποιήθηκαν. Η προσέγγιση χωρίστηκε σε δύο ξεχωριστές φάσεις: δοκιμές API backend και δοκιμές λειτουργικότητας end-to-end της ολοκληρωμένης εφαρμογής.

6.1. Δοκιμές API backend

Το κύριο εργαλείο για την επικύρωση του backend ήταν το **Postman**, μια πλατφόρμα API που αποτελεί πρότυπο στον κλάδο [25]. Πριν και κατά τη διάρκεια της ανάπτυξης του frontend, κάθε τελικό σημείο RESTful API δοκιμάστηκε συστηματικά για να επαληθευτεί η συμπεριφορά και η ασφάλειά του. Δημιουργήθηκε μια ολοκληρωμένη συλλογή Postman για να καλύψει τα ακόλουθα σενάρια για κάθε τελικό σημείο:

- **Θετικές περιπτώσεις δοκιμών:** Αυτές οι δοκιμές επιβεβαίωσαν ότι ένα endpoint επέστρεψε τον αναμενόμενο κωδικό κατάστασης 2xx και ένα σωστά δομημένο JSON payload όταν του δόθηκαν έγκυρα δεδομένα και ένα έγκυρο token αυθεντικοποίησης.
- **Δοκιμές εξουσιοδότησης και ασφάλειας:** Αυτές οι δοκιμές επαλήθευσαν το μοντέλο Role-Based Access Control (RBAC). Τα αιτήματα στάλθηκαν με JWT που ανήκαν σε διαφορετικούς ρόλους χρηστών για να διασφαλιστεί ότι τα endpoints επέστρεφαν σωστά μια κατάσταση 403 Forbidden όταν ένας χρήστης με ανεπαρκή δικαιώματα προσπαθούσε να εκτελέσει μια ενέργεια.
- **Αρνητικές περιπτώσεις δοκιμών:** Αυτές οι δοκιμές επιβεβαίωσαν ότι τα τελικά σημεία επέστρεφαν σωστά τους κατάλληλους κωδικούς σφάλματος 4xx όταν τους παρέχονταν μη έγκυρα δεδομένα (π.χ. ελλείποντα απαιτούμενα πεδία, προκαλώντας αποτυχίες επικύρωσης) ή όταν προσπαθούσαν να αποκτήσουν πρόσβαση σε ανύπαρκτους πόρους (προκαλώντας ένα 404 Not Found).

Αυτή η προσέγγιση δοκιμών API-first ήταν κρίσιμη για τη διασφάλιση της σταθερότητας, της ασφάλειας και της αξιοπιστίας του backend πριν από τη χρήση του από τον frontend client.

6.2. Λειτουργικές δοκιμές από άκρο σε άκρο (end-to-end)

Μόλις ολοκληρώθηκε η ενσωμάτωση του frontend και του backend, πραγματοποιήθηκε μια ολοκληρωμένη φάση χειροκίνητων, λειτουργικών δοκιμών από άκρο σε άκρο. Αυτή η φάση καθοδηγήθηκε από τις **Περιπτώσεις Χρήσης** που ορίζονται στο Κεφάλαιο 2.3. Κάθε κύρια περίπτωση χρήσης για τους τέσσερις καθορισμένους ρόλους (ADMIN, SUPERVISOR, SUPPORT_AGENT, CUSTOMER) εκτελέστηκε από την αρχή έως το τέλος στον browser.

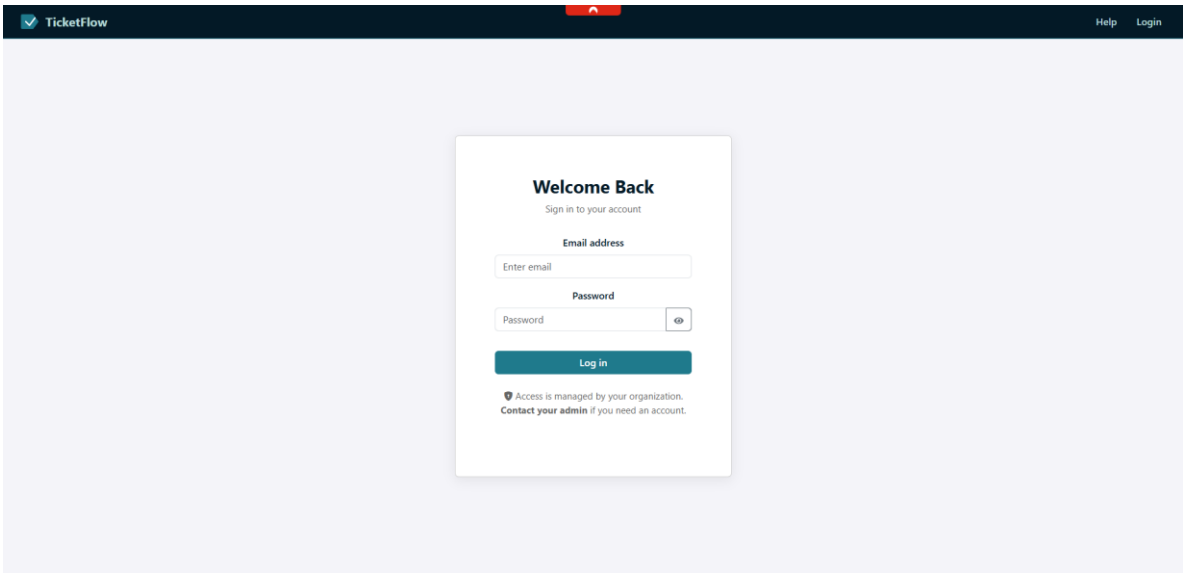
Ο στόχος αυτής της φάσης ήταν να επαληθευτεί ότι το ενσωματωμένο σύστημα λειτουργούσε όπως είχε σχεδιαστεί, ότι το UI αντανακλούσε σωστά την κατάσταση του backend, ότι η συνάρτηση με βάση τους ρόλους στο UI λειτουργούσε και ότι η εφαρμογή ανταποκρινόταν πλήρως σε διαφορετικά μεγέθη οθόνης.

7. Λειτουργικότητες Εφαρμογής

Αυτό το κεφάλαιο παρέχει μια λεπτομερή περιήγηση στις βασικές λειτουργίες της εφαρμογής από την οπτική γωνία των κύριων ρόλων των χρηστών της. Με μια σειρά από στιγμιότυπα οθόνης που συνοδεύονται από περιγραφικό κείμενο, δείχνει πώς ένας χρήστης αλληλεπιδρά με το σύστημα για να επιτύχει τους στόχους του, παρουσιάζοντας την πρακτική εφαρμογή των λειτουργιών που σχεδιάστηκαν στα προηγούμενα κεφάλαια.

7.1. Σελίδα Σύνδεσης

Η πρόσβαση στις βασικές λειτουργίες της εφαρμογής προστατεύεται και απαιτεί έλεγχο ταυτότητας χρήστη. Το κύριο σημείο εισόδου για όλους τους χρήστες που έχουν ταυτοποιηθεί είναι η σελίδα σύνδεσης.



Εικόνα 7.1 Σελίδα Σύνδεσης στο σύστημα

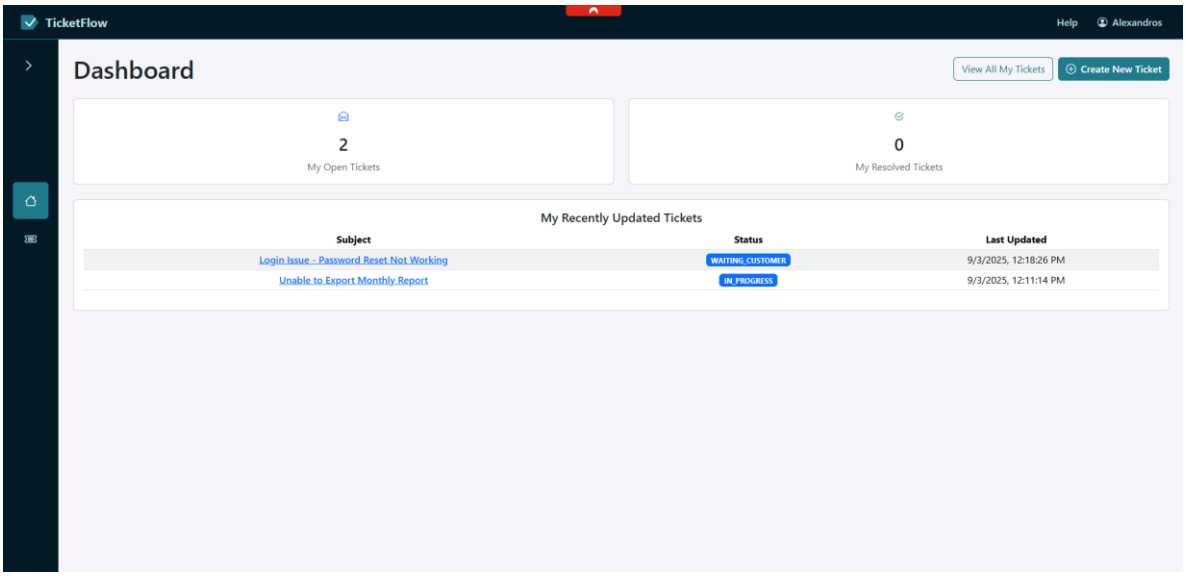
Η Εικόνα 7.1 δείχνει την κύρια διεπαφή σύνδεσης της εφαρμογής. Οι χρήστες ταυτοποιούνται χρησιμοποιώντας το καταχωρημένο email και τον κωδικό πρόσβασής τους. Η φόρμα περιλαμβάνει τυπικές λειτουργίες χρηστικότητας, όπως η δυνατότητα ενεργοποίησης/απενεργοποίησης της ορατότητας του κωδικού πρόσβασης. Μετά την επιτυχή πιστοποίηση, το backend εκδίδει ένα ασφαλές JWT και ο χρήστης ανακατευθύνεται στον πίνακα ελέγχου που αντιστοιχεί στο ρόλο του. Αυτή η σελίδα λειτουργεί ως ασφαλής πύλη πρόσβασης σε όλες τις επόμενες ροές εργασίας.

7.2. Λειτουργικότητες του Πελάτη

Ο ρόλος του Πελάτη (CUSTOMER) έχει σχεδιαστεί ώστε να είναι απλός, διαισθητικός και εστιασμένος, παρέχοντας στους χρήστες τα απαραίτητα εργαλεία για την υποβολή και την παρακολούθηση των αιτημάτων υποστήριξης. Οι ακόλουθες ενότητες απεικονίζουν την πλήρη διαδρομή του χρήστη για έναν πελάτη, από την αρχική προβολή του πίνακα ελέγχου έως τη δημιουργία και την παρακολούθηση ενός αιτήματος υποστήριξης.

7.2.1. Πίνακας Ελέγχου για τον Πελάτη

Μετά την επιτυχή σύνδεση, ο πελάτης μεταφέρεται στον προσωπικό του πίνακα ελέγχου. Αυτή η σελίδα λειτουργεί ως ο κύριος κόμβος δραστηριοτήτων του.



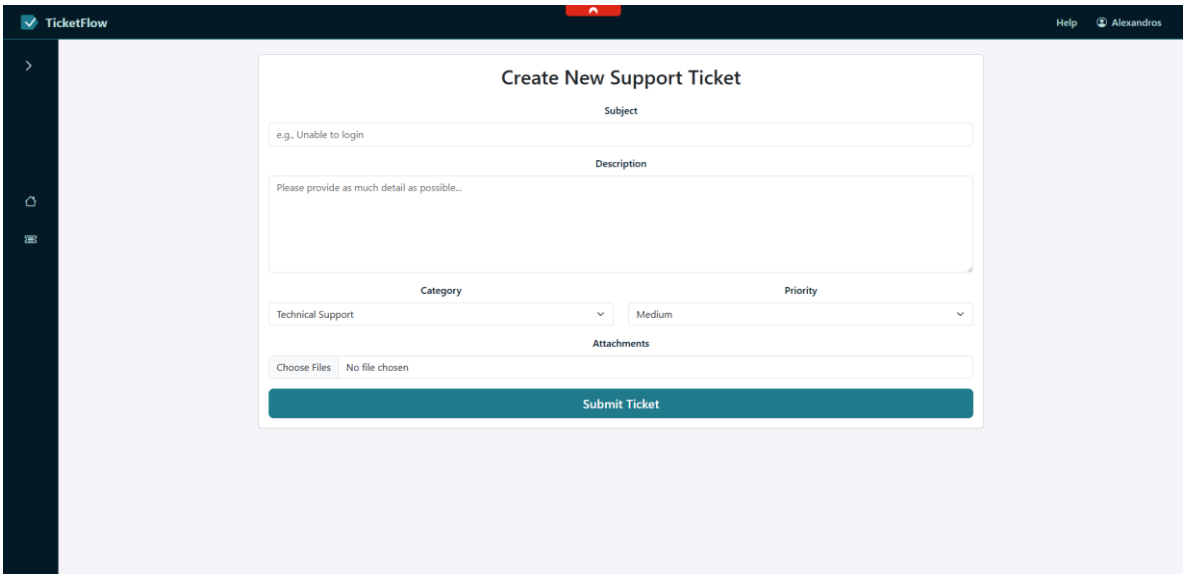
Εικόνα 7.2 Πίνακας Ελέγχου του Πελάτη

Η Εικόνα 7.2 δείχνει τον εξατομικευμένο πίνακα ελέγχου του πελάτη. Αυτή η προβολή έχει σχεδιαστεί για να παρέχει μια άμεση, συνοπτική επισκόπηση της δραστηριότητας υποστήριξης. Οι βασικοί δείκτες απόδοσης (KPI), όπως ο αριθμός των ανοιχτών και επιλυμένων αιτημάτων, εμφανίζονται σε εμφανές σημείο. Κάτω από αυτά, μια λίστα με τα πιο πρόσφατα ενημερωμένα αιτήματά τους παρέχει άμεση πρόσβαση σε τρέχουσες συνομιλίες. Οι κύριες ενέργειες που μπορεί να πραγματοποιήσει ένας πελάτης —δημιουργία νέου αιτήματος ή προβολή του πλήρους ιστορικού αιτημάτων— είναι άμεσα προσβάσιμες μέσω των κουμπιών ενεργειών στην κεφαλίδα.

7.2.2. Δημιουργία Νέου Αιτήματος Υποστήριξης

Η πιο κρίσιμη λειτουργία για έναν πελάτη είναι η δημιουργία ενός νέου αιτήματος υποστήριξης.

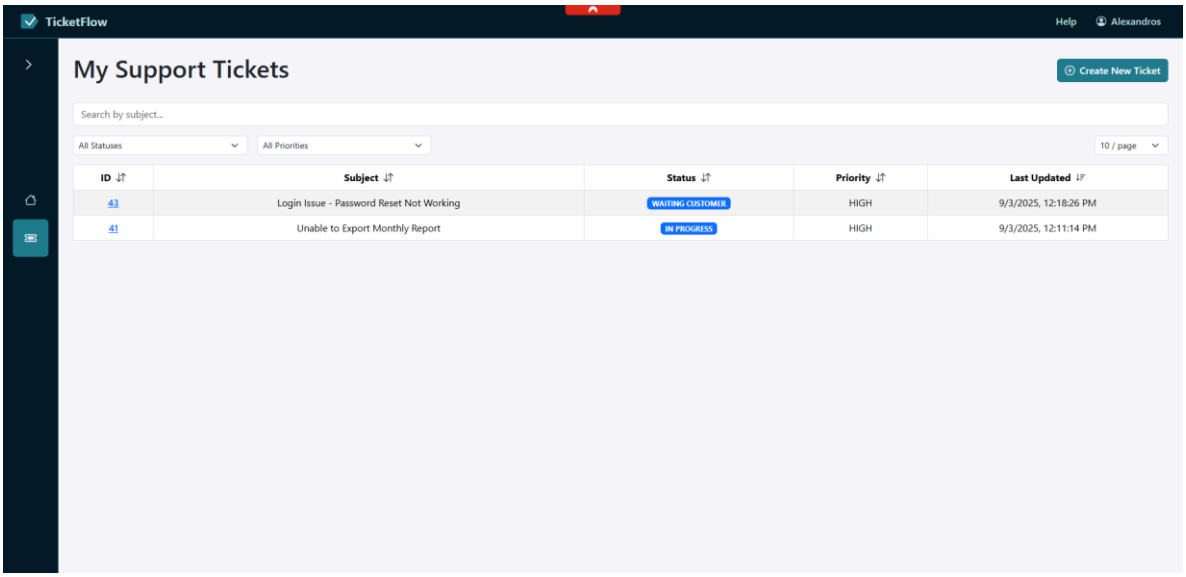
Κάνοντας κλικ στο κουμπί «Create New Ticket», ο χρήστης μεταφέρεται στη φόρμα δημιουργίας εισιτηρίου, όπως φαίνεται στην Εικόνα 7.3. Η φόρμα απαιτεί από τον χρήστη να παρέχει ένα σαφές Θέμα και Περιγραφή και να ταξινομήσει το πρόβλημά του χρησιμοποιώντας τα προκαθορισμένα αναπτυσσόμενα μενού Κατηγορία και Προτεραιότητα. Η διεπαφή υποστηρίζει επίσης την επισύναψη πολλαπλών αρχείων, επιτρέποντας στον χρήστη να παρέχει συμπληρωματικές πληροφορίες, όπως αρχεία καταγραφής ή στιγμιότυπα οθόνης. Τα επιλεγμένα αρχεία εμφανίζονται σε μια λίστα, με την επιλογή να τα αφαιρέσετε πριν από την τελική υποβολή.



Εικόνα 7.3 Η φόρμα δημιουργίας νέου αιτήματος υποστήριξης

7.2.3. Προβολή της Λίστας Αιτημάτων Υποστήριξης του Πελάτη

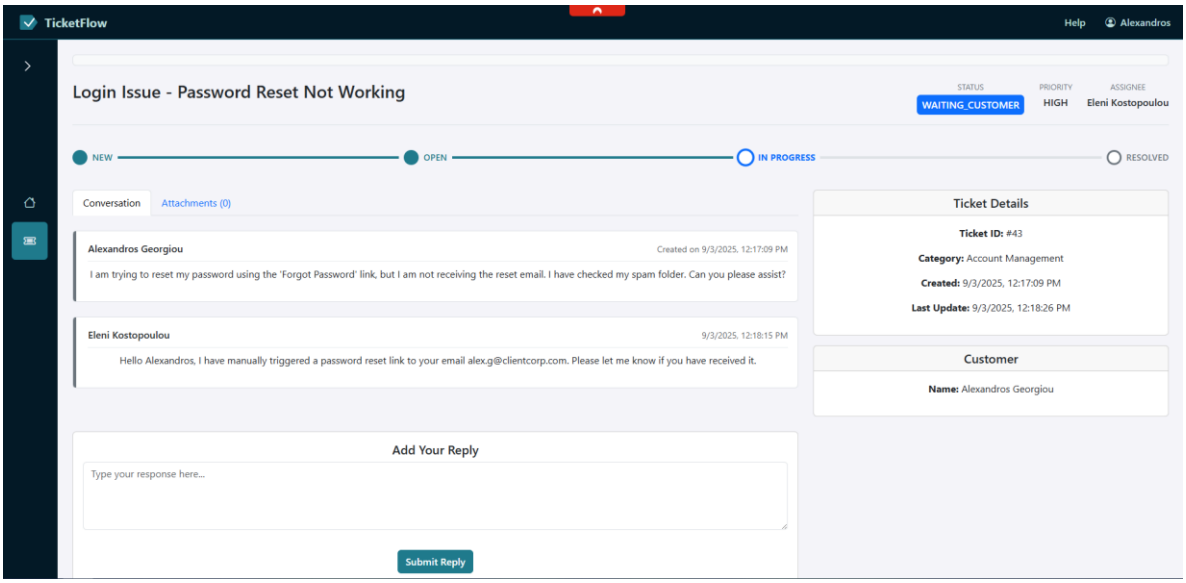
Η σελίδα «My Support Tickets», που απεικονίζεται στην Εικόνα 7.4, παρέχει στον πελάτη μια πλήρη και φιλτραρισμένη λίστα με όλα τα εισιτήρια που έχει υποβάλει. Το πλέγμα δεδομένων έχει συγκεκριμένο πεδίο εφαρμογής για τον πιστοποιημένο χρήστη, εξασφαλίζοντας ότι μπορεί να δει μόνο τα δικά του δεδομένα. Από εδώ, ο χρήστης μπορεί να παρακολουθεί την κατάσταση κάθε αιτήματος και να κάνει κλικ σε οποιοδήποτε αναγνωριστικό εισιτηρίου για να δει το πλήρες ιστορικό του.



Εικόνα 7.4 Η λίστα αιτημάτων υποστήριξης του πελάτη

7.2.4. Προβολή συγκεκριμένου αιτήματος υποστήριξης

Η Εικόνα 7.5 εμφανίζει τη σελίδα Λεπτομέρειες εισιτηρίου από την οπτική γωνία του πελάτη. Η διεπαφή χρήστη έχει σχεδιαστεί με γνώμονα τη σαφήνεια, με μια οπτική ροή επιχειρηματικών διαδικασιών στην κορυφή που υποδεικνύει το τρέχον στάδιο του εισιτηρίου. Ο κύριος πίνακας εμφανίζει το πλήρες, δημόσιο ιστορικό συνομιλιών. Είναι σημαντικό να σημειωθεί ότι τυχόν εσωτερικές σημειώσεις της ομάδας υποστήριξης δεν είναι ορατές στον πελάτη. Ο πελάτης μπορεί να προσθέσει νέες απαντήσεις και συνημμένα, τα οποία θα ανοίξουν αυτόματα το εισιτήριο εάν αυτό βρισκόταν σε αναμονή για απάντησή του.



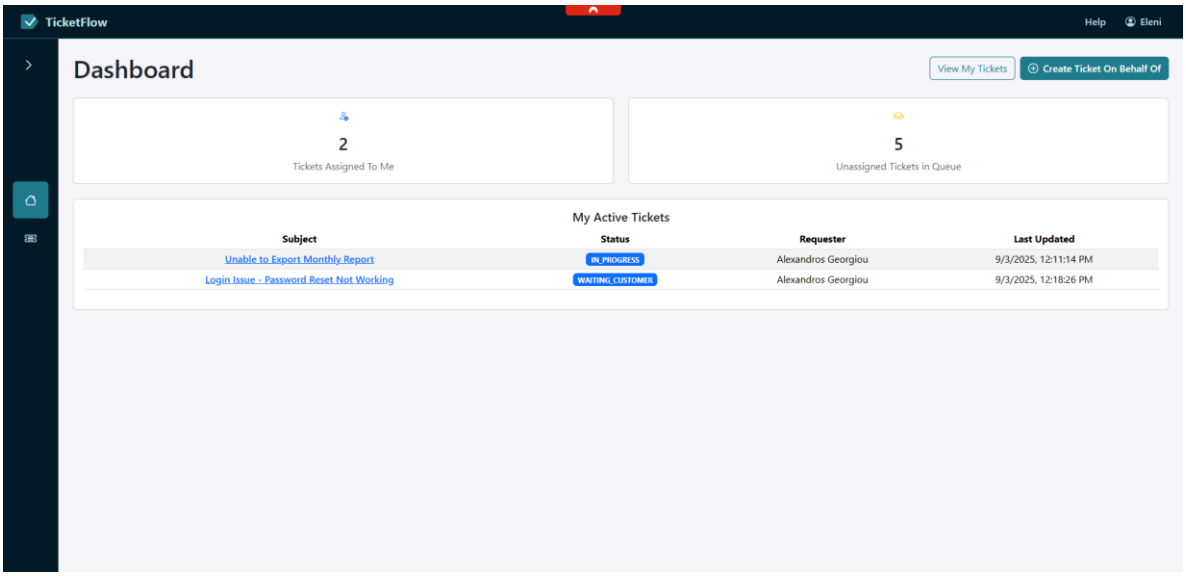
Εικόνα 7.5 Λεπτομέρειες αιτήματος υποστήριξης από την πλευρά του πελάτη

7.3. Λειτουργικότητες του Εκπροσώπου Υποστήριξης

Ο Εκπρόσωπος Υποστήριξης (SUPPORT_AGENT) είναι ο κύριος εσωτερικός χρήστης που είναι υπεύθυνος για την πρακτική επίλυση των προβλημάτων των πελατών. Η διεπαφή χρήστη για αυτόν τον ρόλο έχει σχεδιαστεί ειδικά για να εστιάζει στο προσωπικό φόρτο εργασίας του χρήστη και να απλοποιεί τη διαδικασία επικοινωνίας και διαχείρισης των αιτημάτων.

7.3.1. Πίνακας Ελέγχου για τον Εκπρόσωπο Υποστήριξης

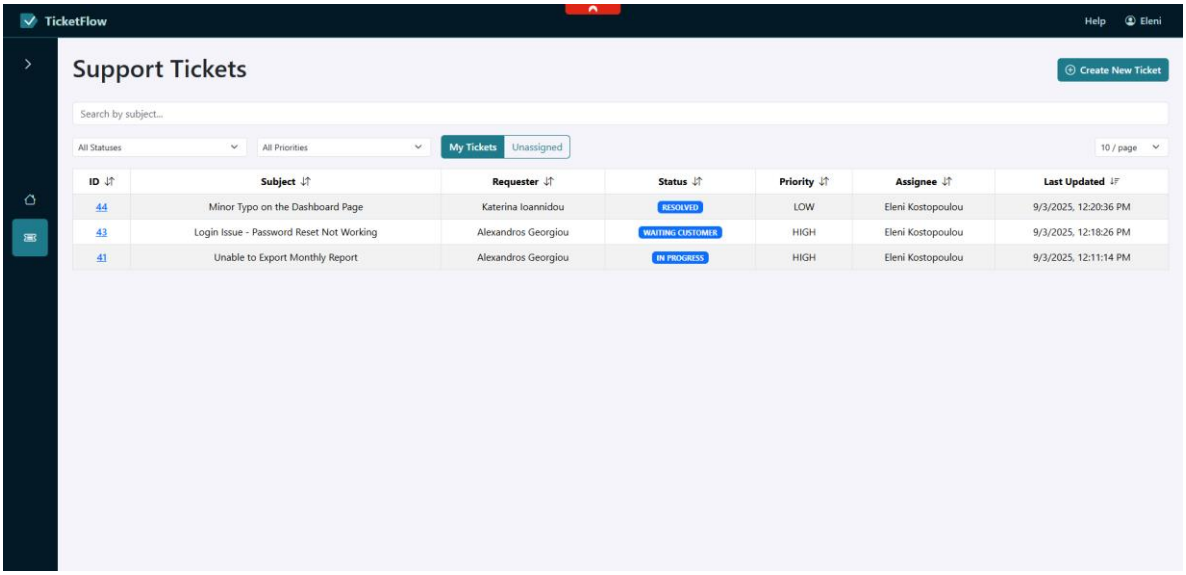
Η Εικόνα 7.6 εμφανίζει τον πίνακα ελέγχου του Support Agent. Αυτή η προβολή είναι προσαρμοσμένη στη ροή εργασίας του πράκτορα, παρουσιάζοντας τις πιο κρίσιμες πληροφορίες για τις καθημερινές του εργασίες. Οι βασικοί δείκτες απόδοσης (KPI) παρέχουν μια γρήγορη σύνοψη του μεγέθους της προσωπικής του ουράς και του αριθμού των μη εκχωρημένων αιτημάτων που είναι διαθέσιμα για διεκδίκηση. Το κύριο widget «My Active Tickets» λειτουργεί ως δυναμική λίστα υποχρεώσεων, ιεραρχώντας τα εισιτήρια κατά προτεραιότητα ανάλογα με την επείγουσα ανάγκη και την τελευταία ενημέρωση. Τα κουμπιά δράσης στην κεφαλίδα παρέχουν άμεσες συντομεύσεις προς την πλήρη λίστα εισιτηρίων και προς τη φόρμα δημιουργίας εισιτηρίου εκ μέρους ενός πελάτη.



Εικόνα 7.6 Πίνακας ελέγχου για τον Υπάλληλο Εξυπηρέτησης

7.3.2. Λίστα Αιτημάτων Υποστήριξης για τον Εκπρόσωπο Εξυπηρέτησης

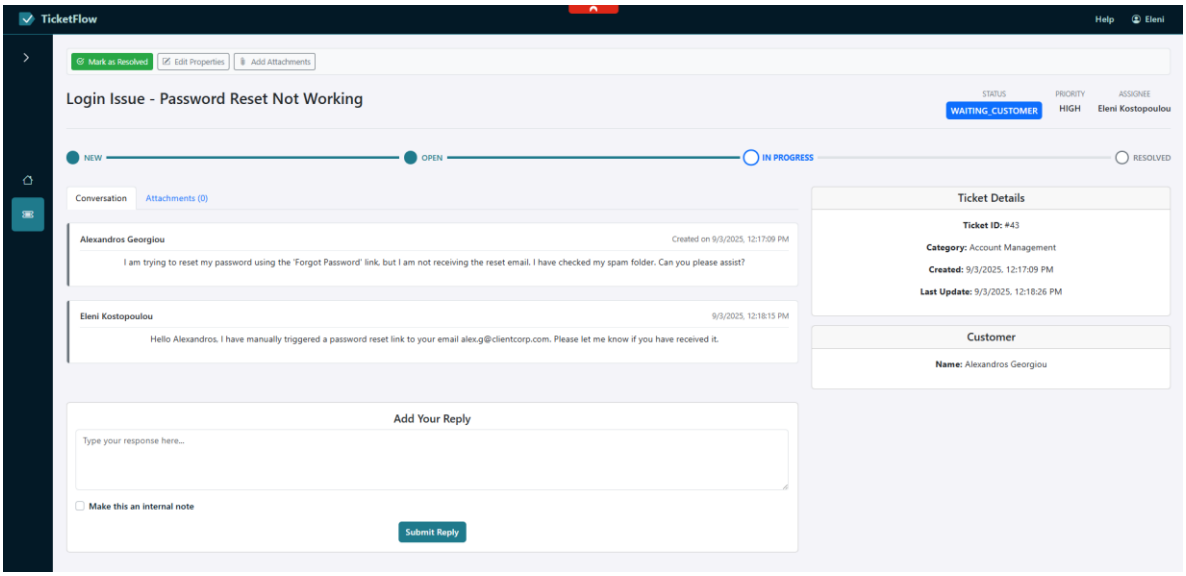
Η προβολή του πράκτορα για τη λίστα των κύριων εισιτηρίων (Εικόνα 7.7) επικεντρώνεται στην ουρά του. Η γραμμή φίλτρου είναι απλοποιημένη και παρέχει ένα κουμπί για εναλλαγή μεταξύ των εισιτηρίων που έχουν ανατεθεί επί του παρόντος σε αυτόν («Τα εισιτήριά μου») και της κοινής ουράς των «Μη ανατεθέντων» εισιτηρίων. Αυτό επιτρέπει στον πράκτορα να διαχειρίζεται αποτελεσματικά το δικό του φόρτο εργασίας ή να διεκδικεί νέα εισιτήρια όταν έχει τη δυνατότητα.



Εικόνα 7.7 Λίστα Αιτημάτων Υποστήριξης για τον Εκπρόσωπο Εξυπηρέτησης

7.3.3. Πληροφορίες Αιτήματος Υποστήριξης

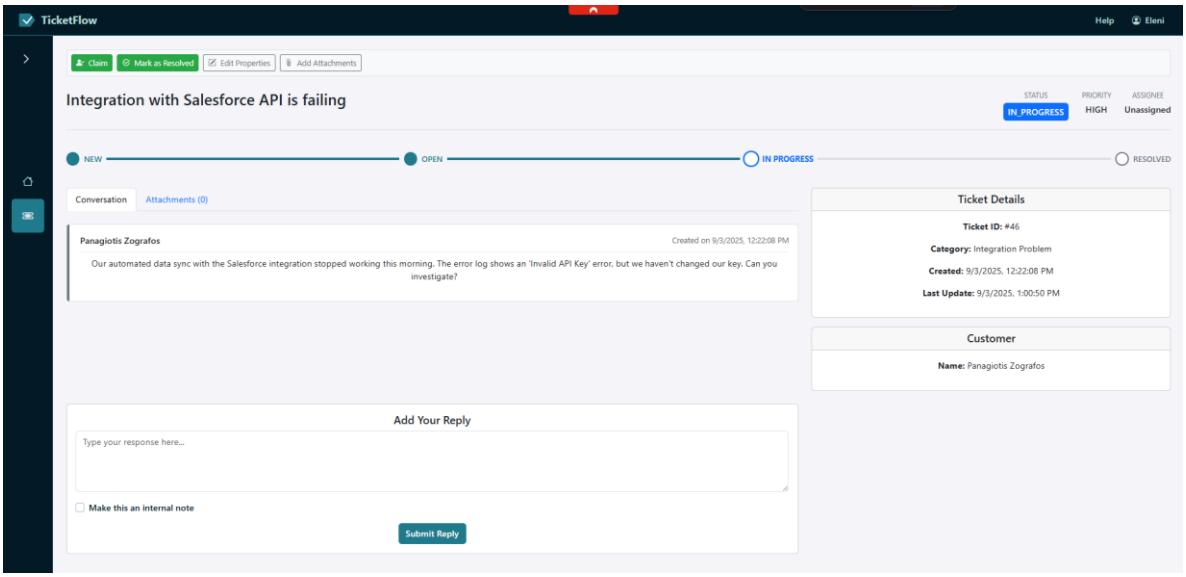
Η σελίδα Λεπτομέρειες εισιτηρίου, που εμφανίζεται στην Εικόνα 7.8, είναι ο κύριος χώρος εργασίας του πράκτορα. Μια γραμμή ενεργειών με αναγνώριση περιβάλλοντος στο επάνω μέρος παρέχει άμεση πρόσβαση στις πιο συνηθισμένες ενέργειες ροής εργασίας, όπως η επίλυση του εισιτηρίου. Ο κύριος πίνακας εμφανίζει το πλήρες ιστορικό συνομιλιών, συμπεριλαμβανομένων των ιδιωτικών «Εσωτερικών σημειώσεων» (που διακρίνονται οπτικά με κίτρινο χρώμα) που χρησιμοποιούνται για τη συνεργασία με άλλα μέλη της ομάδας. Ο πράκτορας μπορεί να προσθέσει νέες δημόσιες απαντήσεις ή εσωτερικές σημειώσεις χρησιμοποιώντας τη φόρμα στο κάτω μέρος.



Εικόνα 7.8 Λεπτομέρειες Αιτήματος Υποστήριξης από την πλευρά του Εκπρόσωπου Εξυπηρέτησης

7.3.4. Διεκδίκηση (Claim) Αιτήματος Υποστήριξης

Η Εικόνα 7.9 απεικονίζει τη διαδικασία με την οποία ένας πράκτορας αναλαμβάνει την ευθύνη για ένα νέο ζήτημα. Όταν προβάλλεται ένα μη εκχωρημένο εισιτήριο, το σύστημα εμφανίζει ένα κουμπί «Απόκτηση» στη γραμμή ενεργειών. Κάνοντας κλικ σε αυτό το κουμπί, το εισιτήριο εκχωρείται αυτόματα στον πράκτορα που είναι συνδεδεμένος εκείνη τη στιγμή και η κατάστασή του ενημερώνεται σε OPEN, εξασφαλίζοντας μια σαφή και άμεση μεταβίβαση της ευθύνης. Αυτή η ενέργεια καταγράφεται επίσης στο αρχείο καταγραφής ελέγχου του εισιτηρίου.



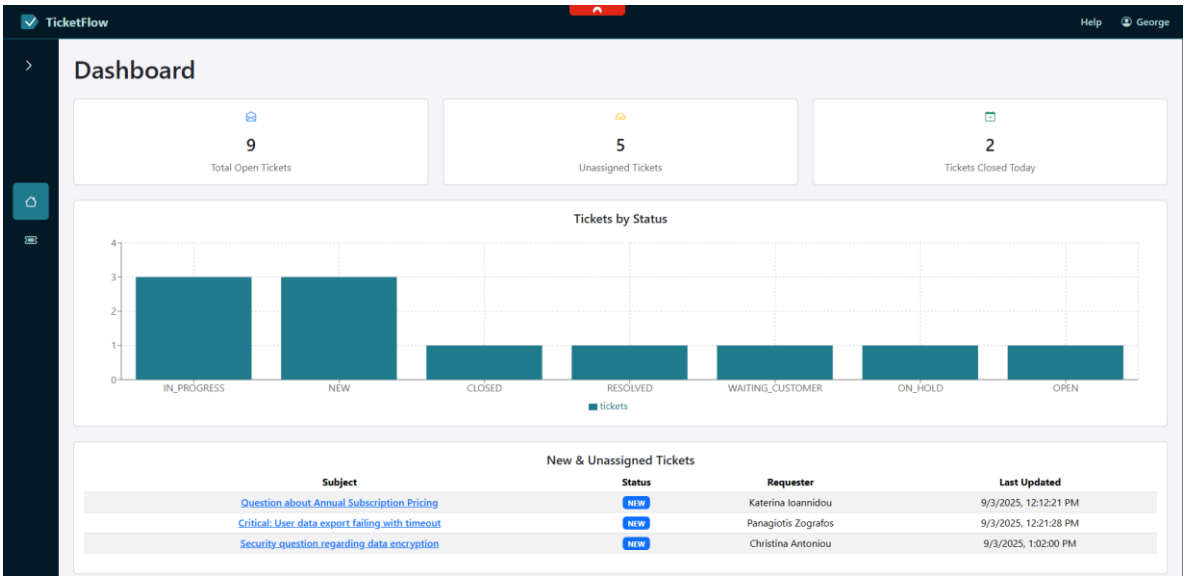
Εικόνα 7.9 Λεπτομέρειες ενός αιτήματος υποστήριξης που δεν είναι ορισμένο σε κάποιον εκπρόσωπο υποστήριξης

7.4. Λειτουργικότητες του Επόπτη

Ο ρόλος του Επόπτη (SUPERVISOR) έχει σχεδιαστεί για τους επικεφαλής ομάδων και τους διευθυντές που είναι υπεύθυνοι για τη συνολική κατάσταση της ουράς υποστήριξης και την απόδοση των υπαλλήλων υποστήριξης. Η διεπαφή τους παρέχει μια συνολική εικόνα υψηλού επιπέδου του συστήματος και των εργαλείων που είναι απαραίτητα για την ταξινόμηση και την ανάθεση των αιτημάτων.

7.4.1. Πίνακας Ελέγχου για τον Επόπτη

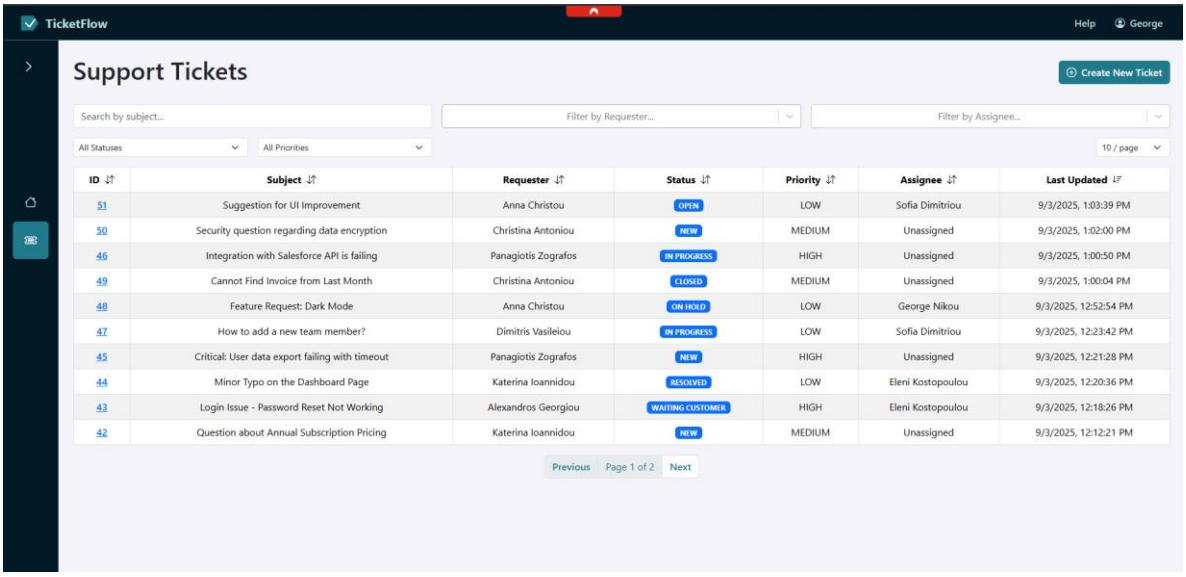
Η Εικόνα 7.10 απεικονίζει τον πίνακα ελέγχου του επόπτη, ο οποίος παρέχει μια ολοκληρωμένη, συνολική εικόνα της κατάστασης του συστήματος υποστήριξης. Σε αντίθεση με την προβολή του πράκτορα, οι βασικοί δείκτες απόδοσης (KPI) αντικατοπτρίζουν τον συνολικό αριθμό των ανοιχτών, μη εκχωρημένων και πρόσφατα κλεισμένων εισιτηρίων σε ολόκληρη την πλατφόρμα. Ένα γράφημα ράβδων απεικονίζει την κατανομή των εισιτηρίων ανάλογα με την τρέχουσα κατάστασή τους, επιτρέποντας την γρήγορη αναγνώριση των σημείων συμφόρησης. Το πιο κρίσιμο στοιχείο είναι η λίστα «New & Unassigned Tickets», η οποία δίνει προτεραιότητα στα παλαιότερα εισιτήρια που περιμένουν ταξινόμηση, επιτρέποντας στον επόπτη να αναλάβει άμεση δράση.



Εικόνα 7.10 Ο πίνακας ελέγχου του Επόπτη

7.4.2. Λίστα αιτημάτων υποστήριξης με φιλτράρισμα

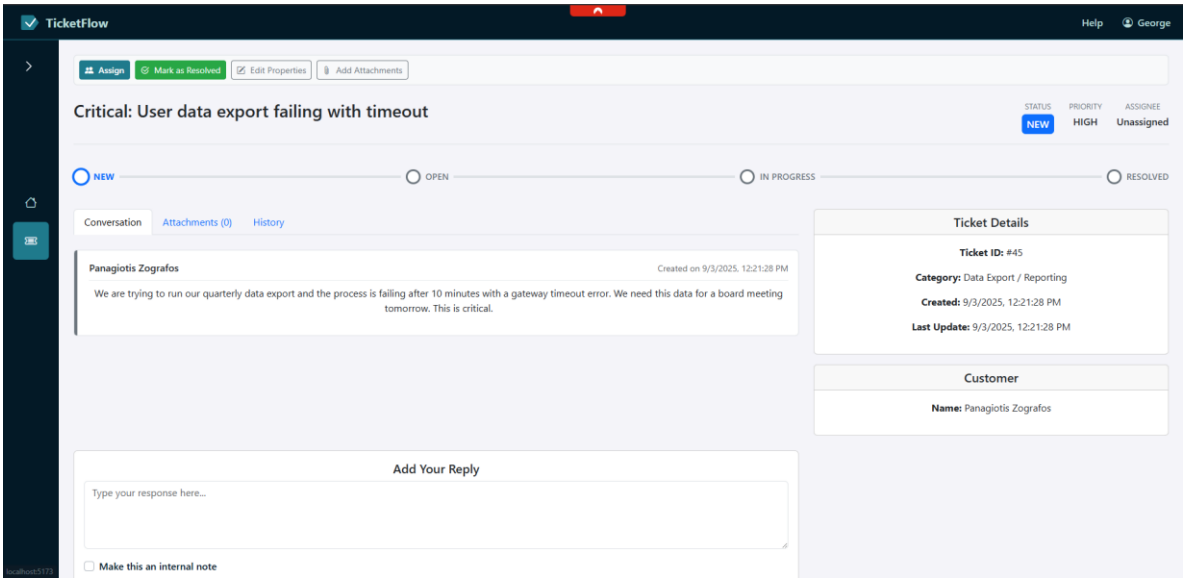
Η προβολή του επόπτη της κύριας λίστας εισιτηρίων, που εμφανίζεται στην Εικόνα 7.11, παρέχει απεριόριστη πρόσβαση σε όλα τα εισιτήρια του συστήματος. Η γραμμή φίλτρου έχει βελτιωθεί με ισχυρά στοιχεία ελέγχου, συμπεριλαμβανομένων αναπτυσσόμενων μενού αναζήτησης για το φιλτράρισμα της λίστας ανά συγκεκριμένο πελάτη (requester) ή ανά συγκεκριμένο μέλος του εσωτερικού προσωπικού (assignee). Αυτή η λειτουργικότητα είναι απαραίτητη για την παρακολούθηση του φόρτου εργασίας των μεμονωμένων πρακτόρων και για τον εντοπισμό συγκεκριμένων εισιτηρίων σε ολόκληρη τη βάση δεδομένων.



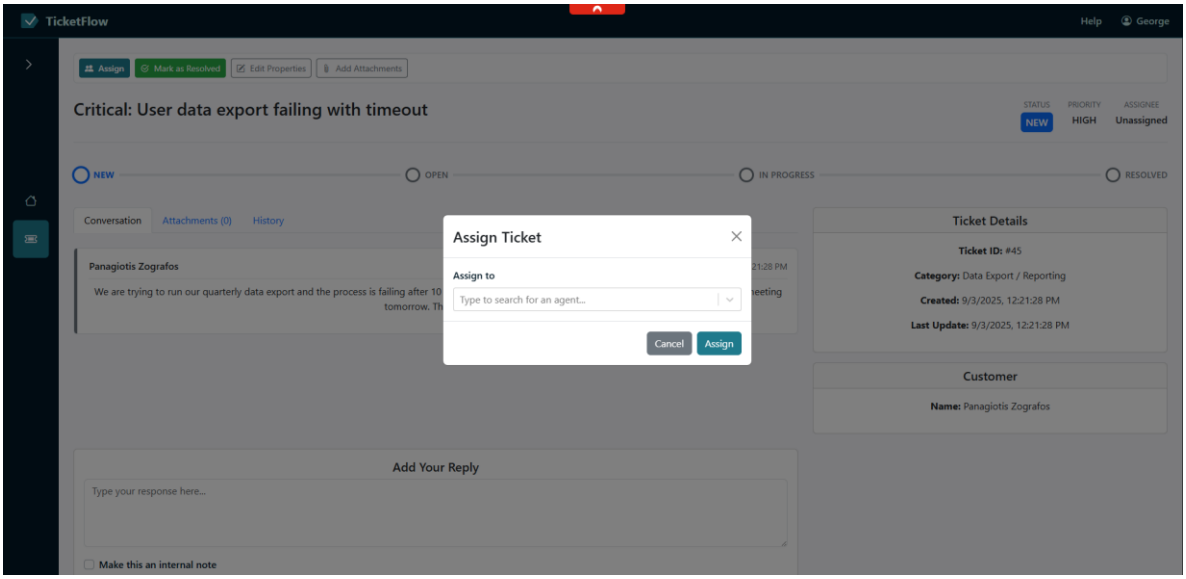
Εικόνα 7.11 Η λίστα των αιτημάτων υποστήριξης μαζί με τα φίλτρα που έχει ο επόπτης διαθέσιμα

7.4.3. Ανάθεση αιτήματος υποστήριξης

Μια από τις κύριες αρμοδιότητες του Προϊσταμένου είναι η ταξινόμηση και η ανάθεση νέων εισιτηρίων. Όπως φαίνεται στην Εικόνα 7.12, όταν προβάλλει ένα εισιτήριο που δεν έχει ανατεθεί, ο Προϊστάμενος μπορεί να κάνει κλικ στο κουμπί «Assign» στη γραμμή ενεργειών για να ανοίξει το παράθυρο ανάθεσης. Αυτό το παράθυρο περιλαμβάνει μια δυναμική, αναζητήσιμη λίστα όλων των μελών του εσωτερικού προσωπικού. Αφού επιλέξει έναν πράκτορα, το εισιτήριο ανατίθεται σε αυτόν, η κατάστασή του ενημερώνεται αυτόματα σε OPEN και η ενέργεια καταγράφεται στο ιστορικό ελέγχου του εισιτηρίου.



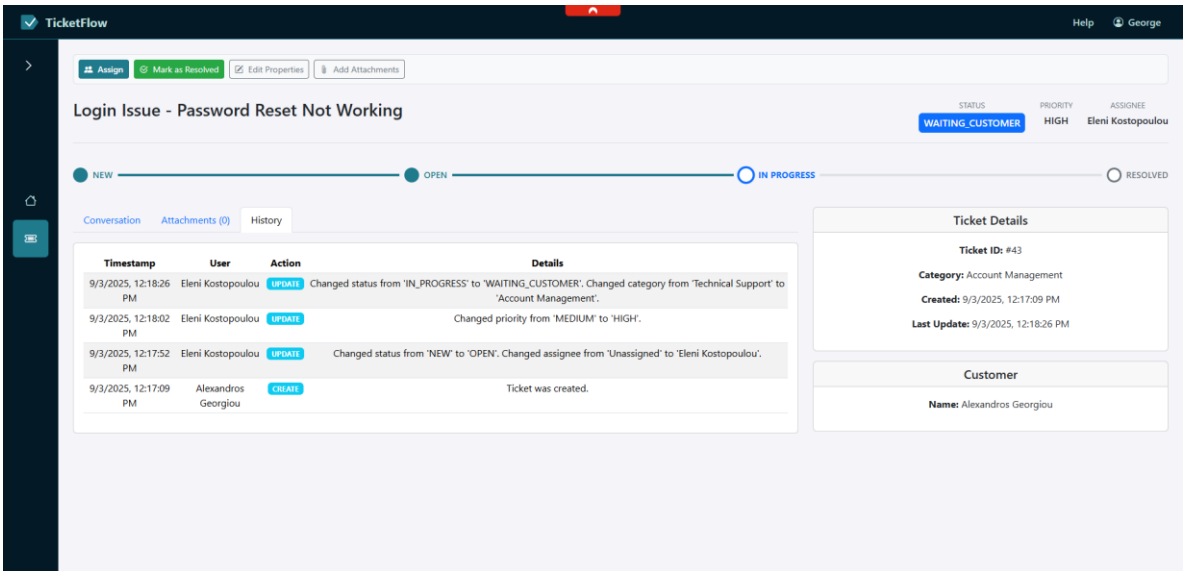
Εικόνα 7.12 Καρτέλα Αιτήματος από την πλευρά του Επόπτη



Εικόνα 7.13 Ανάθεση αιτήματος υποστήριξης από επόπτη

7.4.4. Προβολή του ιστορικού Audit

Για να εξασφαλιστεί η πλήρης λογοδοσία και διαφάνεια, οι επόπτες έχουν πρόσβαση στην καρτέλα «History» για κάθε αίτημα υποστήριξης, όπως φαίνεται στην Εικόνα 7.14. Αυτή η προβολή παρουσιάζει ένα πλήρες, αμετάβλητο ιστορικό ελέγχου για κάθε σημαντική αλλαγή κατάστασης που έχει υποστεί το εισιτήριο, συμπεριλαμβανομένης της δημιουργίας, των αλλαγών κατάστασης και των ανακατανομών. Κάθε καταχώριση στο αρχείο καταγραφής καταγράφει τη χρονική σήμανση, την ενέργεια που εκτελέστηκε, μια λεπτομερή περιγραφή της αλλαγής και τον χρήστη (ή το σύστημα) που είναι υπεύθυνος για την ενέργεια.



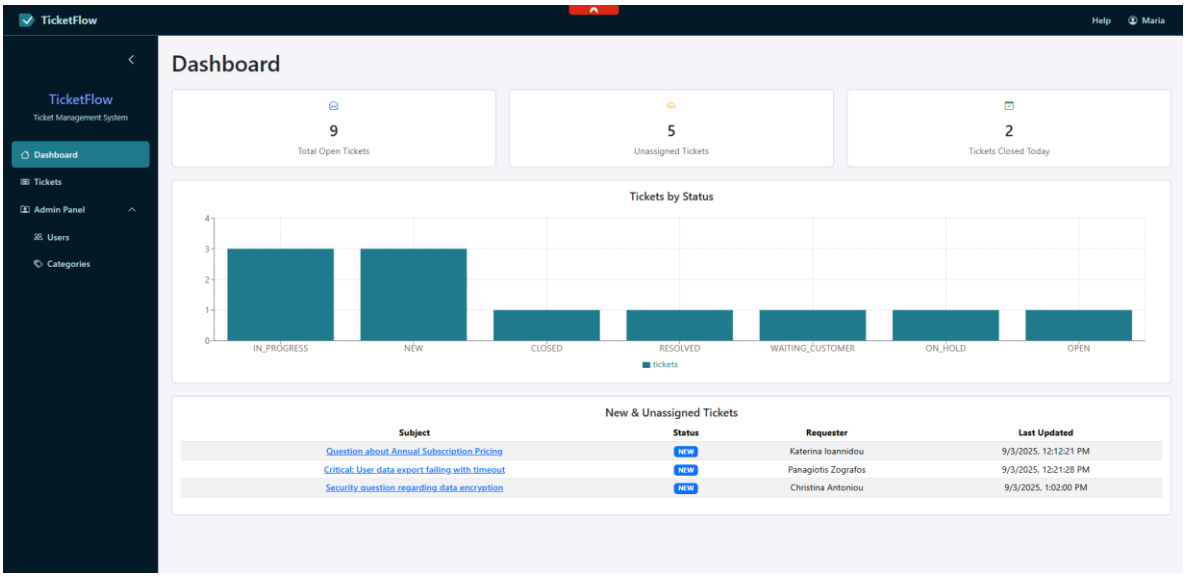
Εικόνα 7.14 Καρτέλα Ιστορικού του Αιτήματος Υποστήριξης

7.5. Λειτουργικότητες του Διαχειριστή

Ο Διαχειριστής (Admin) διαθέτει το υψηλότερο επίπεδο δικαιωμάτων και είναι υπεύθυνος για τη συνολική ρύθμιση, συντήρηση και διαχείριση χρηστών της εφαρμογής. Η παρακάτω ροή εργασίας απεικονίζει τις βασικές διοικητικές λειτουργίες που είναι αποκλειστικές για αυτόν τον ρόλο.

7.5.1. Οθόνη του διαχειριστή

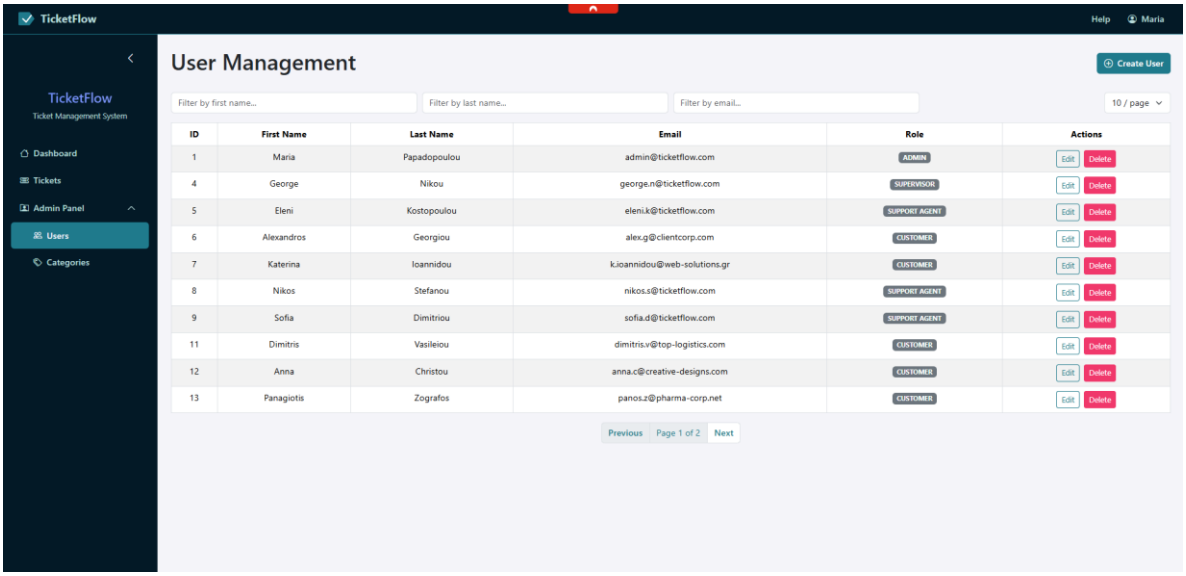
Η Εικόνα 7.15 δείχνει την κύρια προβολή για έναν Διαχειριστή. Ενώ ο πίνακας ελέγχου παρέχει τις ίδιες δυνατότητες παρακολούθησης υψηλού επιπέδου με τον ρόλο του Επόπτη, το βασικό στοιχείο που τον διαφοροποιεί είναι η παρουσία του «Admin Panel» στην γραμμή πλοήγησης. Αυτός ο πίνακας λειτουργεί ως κεντρικός κόμβος πλοήγησης για όλες τις εργασίες διαχείρισης σε επίπεδο συστήματος.



Εικόνα 7.15 Η οθόνη του διαχειριστή με την πλοήγηση στα αριστερά να είναι ανεπτυγμένη

7.5.2. Διαχείριση Χρηστών

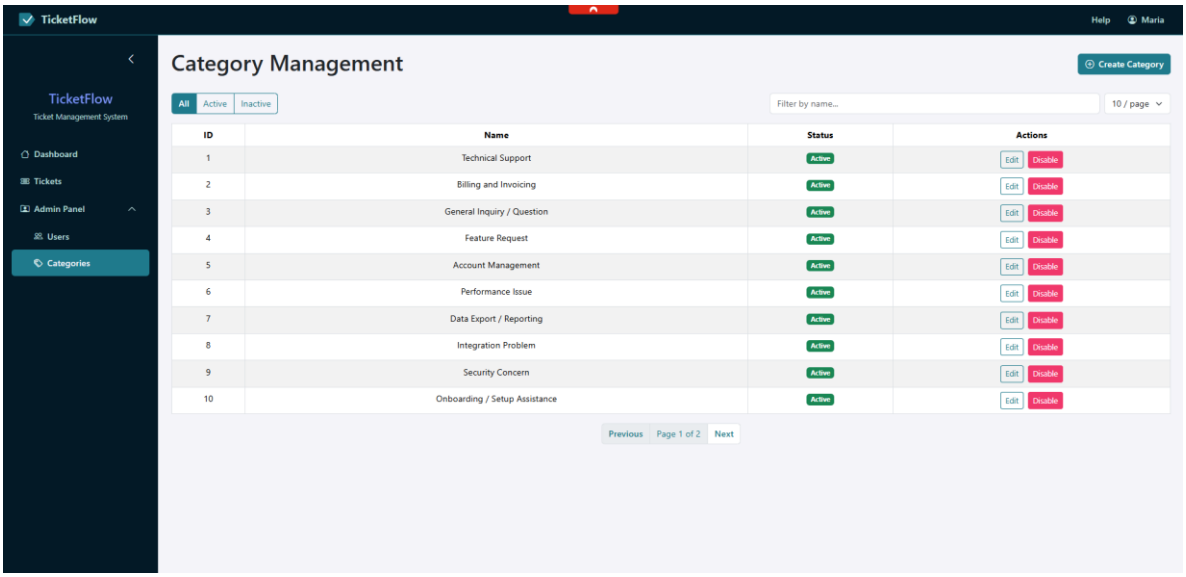
Η σελίδα Διαχείριση χρηστών, που εμφανίζεται στην Εικόνα 7.16, παρέχει στον Διαχειριστή ένα πλήρες πλέγμα δεδομένων για όλους τους χρήστες του συστήματος. Η διεπαφή είναι εξοπλισμένη με λειτουργίες φιλτραρίσματος, ταξινόμησης και σελιδοποίησης από την πλευρά του διακομιστή, για την αποτελεσματική διαχείριση μεγάλου αριθμού χρηστών. Από αυτήν την οθόνη, ο διαχειριστής μπορεί να ξεκινήσει τη δημιουργία ενός νέου χρήστη ή να προχωρήσει στην επεξεργασία ή διαγραφή ενός υπάρχοντος λογαριασμού.



Εικόνα 7.16 Διαχείριση των χρηστών του συστήματος

7.5.3. Διαχείριση Κατηγοριών

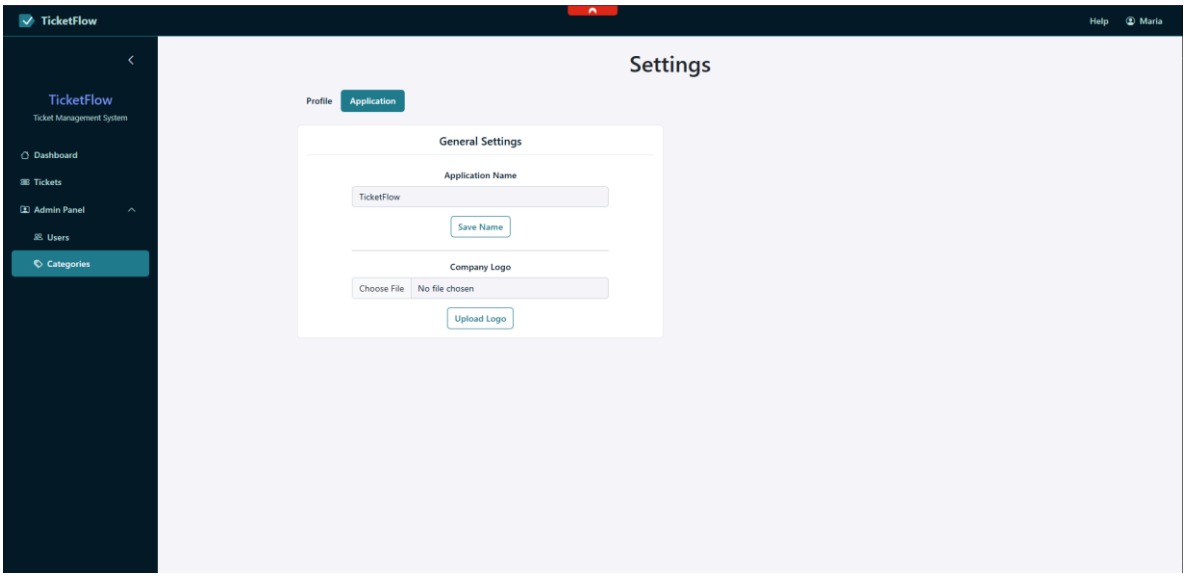
Η διεπαφή Διαχείρισης Κατηγοριών (Εικόνα 7.17) επιτρέπει στον Διαχειριστή να ελέγχει τον κύκλο ζωής των κατηγοριών των αιτημάτων υποστήριξης. Οι διαχειριστές μπορούν να δημιουργούν νέες κατηγορίες, να επεξεργάζονται υπάρχοντα ονόματα και, κυρίως, να απενεργοποιούν κατηγορίες που δεν χρησιμοποιούνται πλέον. Αυτή η λειτουργία «ήπιας διαγραφής» αρχειοθετεί την κατηγορία, κρύβοντάς την από την επιλογή σε νέα αιτήματα, ενώ την διατηρεί στη βάση δεδομένων για να διατηρηθεί η ακεραιότητα των ιστορικών αναφορών.



Εικόνα 7.17 Διαχείριση των κατηγοριών των αιτημάτων

7.5.4. Ρυθμίσεις Εφαρμογής

Η σελίδα Settings παρέχει στον διαχειριστή εργαλεία για τη δυναμική επωνυμία της πλατφόρμας. Όπως φαίνεται στην Εικόνα 7.18, ο διαχειριστής μπορεί να αλλάξει το όνομα της εφαρμογής και να ανεβάσει ένα προσαρμοσμένο λογότυπο. Αυτές οι ρυθμίσεις αποθηκεύονται στη βάση δεδομένων και αντικατοπτρίζονται σε ολόκληρη την εφαρμογή, για παράδειγμα, στην κύρια γραμμή πλοήγησης. Αυτή η λειτουργία επιτρέπει την εύκολη προσαρμογή της εφαρμογής σε διαφορετικά οργανωτικά περιβάλλοντα.



Εικόνα 7.18 Οι ρυθμίσεις εφαρμογής, ορατές μόνο στον διαχειριστή του συστήματος

8. Συμπεράσματα και Μελλοντικές Επεκτάσεις

8.1. Σύνοψη και Συμπεράσματα

Η μεταπτυχιακή αυτή εργασία είχε ως στόχο να αντιμετωπίσει τις σημαντικές λειτουργικές προκλήσεις που υπάρχουν στη διαχείριση των ροών εργασίας υποστήριξης πελατών μέσω του σχεδιασμού και της υλοποίησης μιας πλήρους εφαρμογής ιστού. Ο πρωταρχικός στόχος ήταν η ανάπτυξη ενός ασφαλούς, επεκτάσιμου και διαισθητικού συστήματος διαχείρισης αιτημάτων υποστήριξης, ξεπερνώντας τους περιορισμούς των μη δομημένων εργαλείων όπως το ηλεκτρονικό ταχυδρομείο και τα υπολογιστικά φύλλα. Ο στόχος αυτός επιτεύχθηκε με επιτυχία μέσω της δημιουργίας μιας ολοκληρωμένης πλατφόρμας με πλούσιες λειτουργίες.

Το τελικό σύστημα που υλοποιήθηκε παρέχει μια ισχυρή λύση, βασισμένη σε μια σύγχρονη, αποσυνδεδεμένη **αρχιτεκτονική τριών επιπέδων**. Το backend, που αναπτύχθηκε με το πλαίσιο **Spring Boot**, διαθέτει ένα ασφαλές, stateless **RESTful API** που προστατεύεται από ένα υβριδικό μοντέλο **JSON Web Token (JWT)**, το οποίο περιλαμβάνει έναν μηχανισμό ανάκλησης token για ενισχυμένη ασφάλεια. Η επιχειρηματική λογική του backend είναι εξαιρετικά δομημένη, χρησιμοποιώντας μια πολυεπίπεδη αρχιτεκτονική και το πρότυπο Specification για δυναμικά, ασφαλή requests δεδομένων. Ένα βασικό τεχνικό επίτευγμα του backend είναι το αποσυνδεδεμένο, **event-driven auditing system**, το οποίο παρέχει ένα ασφαλές από συναλλακτική άποψη και πλήρες ιστορικό όλων των αλλαγών κατάστασης των αιτήσεων χωρίς να θέτει σε κίνδυνο την απόδοση ή τη σταθερότητα της βασικής εφαρμογής.

Το frontend αναπτύχθηκε ως μια ανταποκρινόμενη **Single-Page Application (SPA)** χρησιμοποιώντας τη βιβλιοθήκη **React**. Η διεπαφή χρήστη είναι κατασκευασμένη από μια σειρά επαναχρησιμοποιήσιμων στοιχείων και παρέχει μια εξατομικευμένη εμπειρία για καθέναν από τους τέσσερις διαφορετικούς ρόλους χρηστών. Προηγμένες λειτουργίες, όπως σελιδοποίηση από την πλευρά του διακομιστή, φιλτράρισμα χωρίς αναπήδηση και δυναμικά ταμπλό με αναγνώριση ρόλων, υλοποιήθηκαν για να παρέχουν μια επαγγελματική και αποτελεσματική εμπειρία χρήστη.

Συμπερασματικά, αυτή η διατριβή πέτυχε όλους τους αρχικούς της στόχους. Αποτελεί μια πρακτική, ολοκληρωμένη επίδειξη της εφαρμογής σύγχρονων αρχών μηχανικής λογισμικού και αρχιτεκτονικών προτύπων για την επίλυση ενός σύνθετου, πραγματικού επιχειρηματικού προβλήματος. Η εφαρμογή που προέκυψε δεν είναι μόνο ένα λειτουργικό πρωτότυπο, αλλά και μια ισχυρή και επεκτάσιμη βάση που επικυρώνει τις αρχιτεκτονικές αποφάσεις που ελήφθησαν κατά τη φάση του σχεδιασμού. Ο σαφής διαχωρισμός των αρμοδιοτήτων μεταξύ του backend API-first και του reactive frontend, σε συνδυασμό με το λεπτομερές μοντέλο Role-Based Access Control, παρέχει μια ασφαλή και συντηρήσιμη πλατφόρμα για την αποτελεσματική διαχείριση των αιτήσεων υποστήριξης.

8.2. Περιορισμοί της Υλοποίησης

Αν και το σύστημα που υλοποιήθηκε ανταποκρίνεται με επιτυχία σε όλους τους πρωταρχικούς στόχους που τέθηκαν για την παρούσα διατριβή, είναι σημαντικό να αναγνωριστούν οι περιορισμοί του σχεδιασμού του, οι οποίοι έγιναν αποδεκτοί ως αναγκαίες συμβιβαστικές λύσεις για τη διατήρηση ενός εστιασμένου και εφικτού πεδίου εφαρμογής του project. Αυτοί οι περιορισμοί παρέχουν σαφείς κατευθύνσεις για τη μελλοντική ανάπτυξη.

- **Τοπική αποθήκευση αρχείων:** Η τρέχουσα υλοποίηση αποθηκεύει όλα τα συνημμένα που έχουν μεταφορτωθεί στο τοπικό σύστημα αρχείων του διακομιστή. Αν και λειτουργική για ένα

πρωτότυπο με έναν μόνο διακομιστή, αυτή η προσέγγιση δεν είναι οριζόντια επεκτάσιμη. Σε ένα περιβάλλον παραγωγής με πολλαπλές περιπτώσεις διακομιστών ή αναπτύξεις βασισμένες σε κοντέινερ (όπως Docker ή Kubernetes), αυτό θα οδηγούσε σε ασυνέπειες, καθώς ένα αρχείο που έχει μεταφορτωθεί σε μία περίπτωση δεν θα ήταν διαθέσιμο σε άλλη. Μια πιο ισχυρή, εγγενής λύση cloud θα περιελάμβανε την ενσωμάτωση μιας ειδικής υπηρεσίας αποθήκευσης αντικειμένων, όπως το **Amazon S3** ή το **Azure Blob Storage**.

- **Έλλειψη ειδοποιήσεων σε πραγματικό χρόνο:** Η διεπαφή χρήστη της εφαρμογής βασίζεται σε ένα παραδοσιακό μοντέλο αιτήματος-απάντησης. Ο χρήστης πρέπει να ανανεώσει χειροκίνητα τη σελίδα ή να εκτελέσει μια ενέργεια για να δει νέα εισιτήρια ή σχόλια. Ένα πλήρως εξοπλισμένο σύστημα παραγωγής θα εφαρμόζει ενημερώσεις σε πραγματικό χρόνο χρησιμοποιώντας **WebSockets** (π.χ. μέσω STOMP over Spring). Αυτό θα επιτρέπει στον διακομιστή να στέλνει ζωντανές ενημερώσεις στον πελάτη, ειδοποιώντας αμέσως έναν πράκτορα για ένα νέο εισιτήριο ή έναν πελάτη για μια νέα απάντηση χωρίς να απαιτείται χειροκίνητη ανανέωση.
- **Βασική αναζήτηση χρηστών για μεγάλα σύνολα δεδομένων:** Η ασύγχρονη λειτουργία αναζήτησης για την επιλογή χρηστών (π.χ. στο παράθυρο «Assign Ticket») έχει σχεδιαστεί για την αναζήτηση μικρού έως μεσαίου αριθμού χρηστών. Σε ένα σύστημα με δεκάδες χιλιάδες ή εκατομμύρια πελάτες, αυτό το απλό LIKE query θα μπορούσε να αποτελέσει εμπόδιο στην απόδοση. Μια πιο επεκτάσιμη λύση θα περιλαμβάνει την εφαρμογή ενός ειδικού ευρετηρίου αναζήτησης χρησιμοποιώντας μια τεχνολογία όπως το **Elasticsearch**, το οποίο είναι βελτιστοποιημένο για αναζήτηση πλήρους κειμένου υψηλής ταχύτητας σε τεράστια σύνολα δεδομένων.
- **Στατικοί ορισμοί ροής εργασίας:** Οι καταστάσεις του κύκλου ζωής του εισιτηρίου (π.χ. NEW, OPEN, RESOLVED) ορίζονται επί του παρόντος ως στατικό Java Enum. Αυτή είναι μια ισχυρή και ασφαλής προσέγγιση, αλλά στερείται της ευελιξίας που απαιτείται για να ορίσει μια οργάνωση τις δικές της προσαρμοσμένες ροές εργασίας.

8.3. Προτάσεις για Μελλοντικές Επεκτάσεις

Η τρέχουσα εφαρμογή χρησιμεύει ως μια ισχυρή και επεκτάσιμη βάση. Η αποσυνδεδεμένη αρχιτεκτονική API-first και η αρθρωτή βάση κώδικα παρέχουν πολλές ευκαιρίες για μελλοντικές βελτιώσεις που θα αναβαθμίσουν το σύστημα σε μια πλατφόρμα με πλήρεις λειτουργίες. Τα παρακάτω είναι τα πιο σημαντικά και λογικά επόμενα βήματα για τη μελλοντική ανάπτυξη.

- **Ασύγχρονο σύστημα ειδοποίησης μέσω email:** Ένα κρίσιμο χαρακτηριστικό για οποιοδήποτε σύστημα υποστήριξης είναι η δυνατότητα να ειδοποιεί προληπτικά τους χρήστες για σημαντικά γεγονότα. Η υπάρχουσα αρχιτεκτονική που βασίζεται σε γεγονότα (με TicketCreatedEvent, TicketUpdatedEvent κ.λπ.) είναι απόλυτα κατάλληλη για αυτό. Θα μπορούσε να υλοποιηθεί ένα νέο NotificationEventListener, που θα λειτουργεί ασύγχρονα με το annotation @Async. Αυτός ο ακροατής θα καταναλώνει τα συμβάντα της εφαρμογής και θα χρησιμοποιεί μια ειδική υπηρεσία MailService με μηχανή προτύπων (όπως το Thymeleaf) για να στέλνει επαγγελματικά email HTML στους χρήστες για σημαντικά συμβάντα, όπως ένα νέο σχόλιο στο ticket τους, μια αλλαγή στην κατάσταση σε Resolved ή μια νέα ανάθεση ticket για έναν πράκτορα.
- **Ροές εργασίας που μπορούν να διαμορφωθούν από τον διαχειριστή (δυναμικές καταστάσεις):** Για να παρέχεται μέγιστη ευελιξία, η στατική Status Enum θα μπορούσε να αναδιαμορφωθεί σε μια πλήρη οντότητα JPA. Αυτό θα επέτρεπε σε έναν διαχειριστή να

δημιουργεί, να επεξεργάζεται και να ταξινομεί προσαρμοσμένες καταστάσεις μέσω του UI. Αυτό θα συνδυαζόταν με ένα μοντέλο «State» (π.χ. OPEN, CLOSED), όπως φαίνεται σε πλατφόρμες όπως το Microsoft Dynamics. Ένας διαχειριστής θα μπορούσε να δημιουργήσει μια νέα κατάσταση όπως «Awaiting Dev Team» (Σε αναμονή της dev ομάδας) και να την αντιστοιχίσει στην κατάσταση OPEN, επιτρέποντας στις επιχειρήσεις να προσαρμόσουν τέλεια τον κύκλο ζωής των εισιτηρίων στις συγκεκριμένες εσωτερικές διαδικασίες τους χωρίς να απαιτούνται αλλαγές στον κώδικα.

- **Ενημερώσεις διεπαφής χρήστη σε πραγματικό χρόνο με WebSockets:** Για να δημιουργηθεί μια πιο δυναμική και συνεργατική εμπειρία χρήστη, το σύστημα θα μπορούσε να βελτιωθεί με δυνατότητες σε πραγματικό χρόνο. Με την ενσωμάτωση **WebSockets** (π.χ. χρησιμοποιώντας την υποστήριξη μηνυμάτων STOMP του Spring), ο διακομιστής backend θα μπορούσε να προωθεί ζωντανές ενημερώσεις απευθείας στους πελάτες frontend. Αυτό θα σήμαινε ότι η λίστα εισιτηρίων ενός πράκτορα θα ενημερωνόταν αμέσως όταν ένας επόπτης τους αναθέτει ένα νέο αίτημα, ή ένα thread συνομιλίας θα ενημερωνόταν σε πραγματικό χρόνο για όλους τους θεατές όταν δημοσιεύεται ένα νέο σχόλιο, εξαλείφοντας την ανάγκη για χειροκίνητη ανανέωση της σελίδας.
- **Ενσωμάτωση λύσης αποθήκευσης αρχείων με βάση το cloud:** Για να ξεπεραστούν οι περιορισμοί επεκτασιμότητας της τοπικής αποθήκευσης αρχείων, το FileStorageService θα μπορούσε να αναδιαμορφωθεί ώστε να χρησιμοποιεί μια λύση αποθήκευσης αντικειμένων με βάση το cloud, όπως το **Amazon S3** ή το **Azure Blob Storage**. Αυτό θα παρείχε ένα εξαιρετικά ανθεκτικό, επεκτάσιμο και παγκοσμίως προσβάσιμο backend αποθήκευσης για όλα τα συνημμένα και τα λογότυπα, το οποίο είναι απαραίτητο για μια ανάπτυξη παραγωγής με πολλαπλές περιπτώσεις διακομιστών.
- **Προηγμένες αναλύσεις και αναφορές:** Ο τρέχων πίνακας ελέγχου παρέχει μια γενική επισκόπηση υψηλού επιπέδου. Μια μελλοντική βελτίωση θα ήταν η δημιουργία ενός ειδικού τμήματος «Αναφορών». Αυτό θα μπορούσε να αξιοποιήσει τα ιστορικά δεδομένα των αιτημάτων και των αρχείων καταγραφής ελέγχου για τη δημιουργία πιο σύνθετων αναφορών, όπως ο μέσος χρόνος πρώτης απόκρισης, ο χρόνος επίλυσης ανά πράκτορα και η ανάλυση τάσεων των αιτήσεων ανά κατηγορία με την πάροδο του χρόνου.

Αυτές οι προτεινόμενες βελτιώσεις καταδεικνύουν την επεκτασιμότητα και την ευελιξία της βασικής αρχιτεκτονικής της εφαρμογής, παρέχοντας ένα σαφές σχέδιο δράσης για την εξέλιξή της.

9. Πίνακας ορολογίας

Ελληνικά	Αγγλικά
Σύστημα Διαχείρισης Πελατειακών Σχέσεων	Customer Relationship Management (CRM)
Έλεγχος Πρόσβασης βάσει Ρόλων	Role-Based Access Control (RBAC)
Διεπαφή Προγραμματισμού Εφαρμογών	Application Programming Interface (API)
Εφαρμογή Μονοσέλιδης Αρχιτεκτονικής	Single-Page Application (SPA)
Διαδικτυακό Διακριτικό JSON	JSON Web Token (JWT)
Διάγραμμα Σχέσεων Οντοτήτων	Entity-Relationship Diagram (ERD)
Πλήρης Στοίβα (Τεχνολογιών)	Full-Stack
Προβολή	View
Ελεγκτής	Controller
Υπηρεσία	Service
Αντικειμενική σχεσιακή αντιστοίχιση	Object-relational mapping
Πόροι	Resources
Αντικείμενα μεταφοράς δεδομένων	Data Transfer Objects
Βάση δεδομένων	Database
μηχανισμός καταγραφής ενεργειών	Audit
Διακομιστής	server
Συμβάντα	events
Επισήμανση	annotation
Ακροατής συμβάντων	Event Listener
εκδότης συμβάντων	Event publisher
Αποθετήριο	repository
ευπάθειες μαζικής εκχώρησης	mass assignment vulnerabilities
Έγχυση εξαρτήσεων	Dependency injection

10. Πίνακας συντμήσεων-αρκτικόλεξων-ακρωνυμίων

CRM	Customer Relationship Management
RBAC	Role-Based Access Control
API	Application Programming Interface
SPA	Single-Page Application
JWT	JSON Web Token
JPA	Jakarta Persistence API
MVC	Model-View-Controller
ORM	Object-relational mapping
URI	Uniform Resource Identifier
JSON	JavaScript Object Notation
DTO	Data Transfer Object
DB	Database
POJO	Plain Old Java Object
CRUD	Create, Read, Update, Delete
ACID	Atomicity, Consistency, Isolation, Durability
IoC	Inversion of Control
KPI	Key Performance Indicator

11. Βιβλιογραφία

[1] Francis Buttle and Stan Maklan, “Customer Relationship Management | Concepts and Technologies | Francis.” Accessed: Sept. 01, 2025. [Online]. Available: https://www.taylorfrancis.com/books/mono/10.4324/9781351016551/customer-relationship-management-francis-buttle-stan-maklan

[2] R. S. Pressman, *Software Engineering: A Practitioner’s Approach*. Palgrave Macmillan, 2005.

[3] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, “Proposed NIST standard for role-based access control,” *ACM Trans Inf Syst Secur*, vol. 4, no. 3, pp. 224–274, Aug. 2001, doi: 10.1145/501978.501980.

[4] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2012.

[5] R. T. Fielding, “Architectural styles and the design of network -based software architectures - ProQuest.” Accessed: Sept. 01, 2025. [Online]. Available: https://www.proquest.com/openview/fc2d064044b971dda476dfb429a2b344/1?pq-origsite=gscholar&cbl=18750&diss=y

[6] M. Kleppmann, *Designing data-intensive applications: the big ideas behind reliable, scalable, and maintainable systems*, First edition. Beijing Boston Farnham Sebastopol Tokyo: O’Reilley, 2017.

[7] Atlassian, “Jira Service Management.” Accessed: Sept. 02, 2025. [Online]. Available: https://www.atlassian.com/software/jira/service-management

[8] “Zendesk for Service,” Zendesk. Accessed: Sept. 02, 2025. [Online]. Available: https://www.zendesk.com/service/

[9] Microsoft, “Microsoft Dynamics 365 Customer Service.” Accessed: Sept. 02, 2025. [Online]. Available: https://www.microsoft.com/el-gr/dynamics-365

[10] “HubSpot | Software & Tools for your Business - Homepage.” Accessed: Sept. 10, 2025. [Online]. Available: https://www.hubspot.com

[11] A. Cockburn, “Writing Effective Use Cases,” 2001.

[12] React Team, “React.” Accessed: Sept. 01, 2025. [Online]. Available: https://react.dev/

[13] React Bootstrap Team, “React-Bootstrap.” Accessed: Sept. 03, 2025. [Online]. Available: https://react-bootstrap.netlify.app/

[14] T. Haerder and A. Reuter, “Principles of transaction-oriented database recovery,” *ACM Comput Surv*, vol. 15, no. 4, pp. 287–317, Dec. 1983, doi: 10.1145/289.291.

[15] M. Jones, J. Bradley, and N. Sakimura, *RFC 7519: JSON Web Token (JWT)*. USA: RFC Editor, 2015.

[16] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Deutschland GmbH, 1995.

[17] E. Evans, *Domain-driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, 2004.

[18] E. Sakkopoulos, D. Antoniou, P. Adamopoulou, N. Tsirakis, and A. Tsakalidis, “A web personalizing technique using adaptive data structures: The case of bursts in web visits,” *J. Syst. Softw.*, vol. 83, no. 11, pp. 2200–2210, Nov. 2010, doi: 10.1016/j.jss.2010.06.026.

[19] OWASP, “OWASP Top 10:2021.” Accessed: Aug. 31, 2025. [Online]. Available: https://owasp.org/www-project-top-ten/

[20] M. Fowler, “Inversion of Control Containers and the Dependency Injection pattern,” martinfowler.com. Accessed: Aug. 31, 2025. [Online]. Available: https://martinfowler.com/articles/injection.html

[21] OWASP, “Cross-Site Request Forgery Prevention Cheat Sheet.” Accessed: Sept. 01, 2025. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

[22] “Coolors,” Coolors.co. Accessed: Sept. 03, 2025. [Online]. Available: https://coolors.co/031926-1f7a8c-b8dbd9-f4f4f9-f0386b

[23] Remix Team, “React Router Official Documentation.” Accessed: Sept. 01, 2025. [Online]. Available: https://reactrouter.com/

[24] React Team, “Passing Data Deeply with Context – React.” Accessed: Sept. 01, 2025. [Online]. Available: https://react.dev/learn/passing-data-deeply-with-context
[25] Postman, “Postman API Platform.” Accessed: Sept. 03, 2025. [Online]. Available: https://www.postman.com/

12. Παραρτήματα

Παράρτημα Α: Repository Πηγαίου Κώδικα

Ο πλήρης πηγαίος κώδικας για την εφαρμογή, συμπεριλαμβανομένων τόσο του Spring Boot backend όσο και του React frontend, είναι διαθέσιμος για έλεγχο και ανάλυση στο ακόλουθο δημόσιο GitHub repository:

https://github.com/a-efstathiou/crm

Το repository περιλαμβάνει το πλήρες ιστορικό των commit, το οποίο τεκμηριώνει τη διαδικασία ανάπτυξης από την έναρξη έως την τελική έκδοση που παρουσιάζεται σε αυτή τη διατριβή.