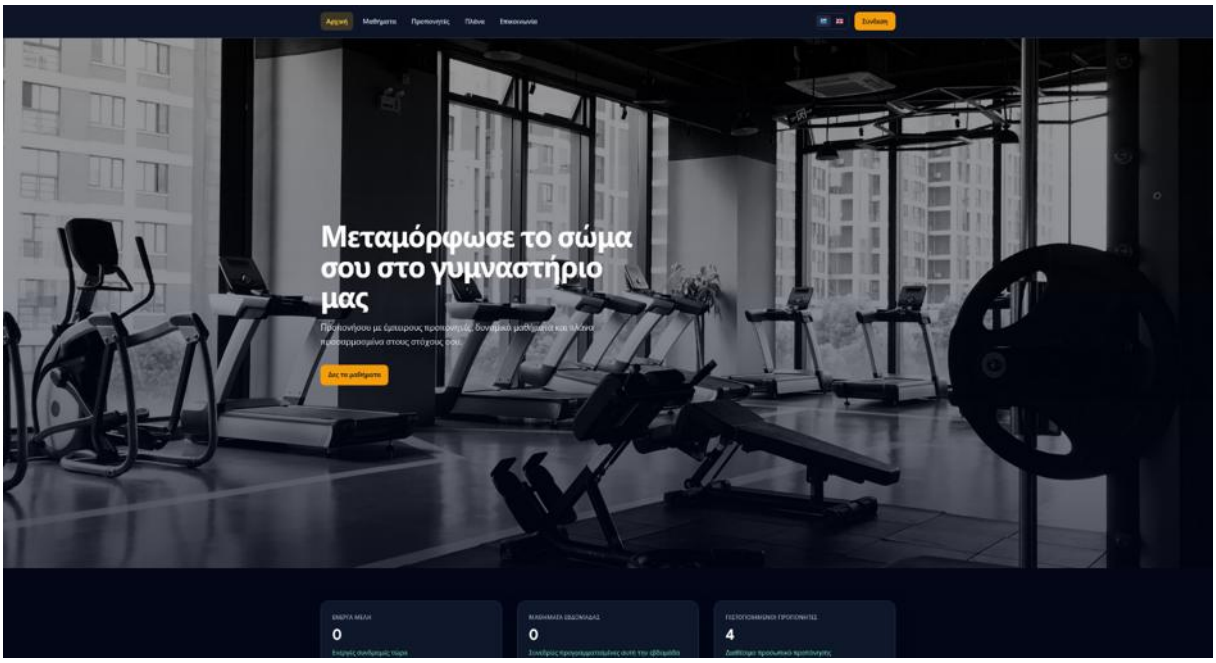
ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Ανάπτυξη διαδικτυακής εφαρμογής διαχείρισης πελατών γυμναστηρίου»



Του φοιτητή
Δήμου Δημήτριο
Αρ. Μητρώου: 2019037

Επιβλέπων
Δρ. Κυριάκος Τσιακμάκης

ΙΟΥΝΙΟΣ 2026

Τίτλος Δ.Ε. **Ανάπτυξη διαδικτυακής εφαρμογής διαχείρισης πελατών γυμναστηρίου**
Κωδικός Δ.Ε. **25145**
Ονοματεπώνυμο φοιτητή. **Δήμου Δημήτριος**
Ονοματεπώνυμο εισηγητή. **Δρ Κυριάκος Τσιακμάκης**
Ημερομηνία ανάληψης Δ.Ε. **06-03-2025**
Ημερομηνία περάτωσης Δ.Ε. **31-05-2026**

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή Δήμου Δημήτριο που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης, ο συγγραφέας/δημιουργός εκχωρεί στο Διεθνές Πανεπιστήμιο της Ελλάδος άδεια χρήσης του δικαιώματος αναπαραγωγής, δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσης της εργασίας διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος. Η ανοικτή πρόσβαση στο πλήρες κείμενο της εργασίας, δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού, ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, πώληση, εμπορική χρήση, διανομή, έκδοση, μεταφόρτωση (downloading), ανάρτηση (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

Πρόλογος

Η επιλογή του θέματος έγινε για να αξιοποιηθούν οι τεχνολογίες που διδάχθηκαν στα προηγούμενα ακαδημαϊκά έτη, σε συνδυασμό με σύγχρονα εργαλεία ανάπτυξης, μέσα από ένα ρεαλιστικό έργο. Η δημιουργία πλατφόρμας διαχείρισης γυμναστηρίου κρίθηκε κατάλληλη, καθώς στην ελληνική πραγματικότητα υπάρχει έλλειψη οργανωμένων ψηφιακών λύσεων, ενώ πολλά γυμναστήρια δεν διαθέτουν καν βασική διαδικτυακή παρουσία.

Το κύριο όφελος της εργασίας ήταν η εφαρμογή της θεωρίας σε πραγματικές συνθήκες ανάπτυξης, με έμφαση στον σωστό σχεδιασμό, την οργάνωση δεδομένων και την πρακτική ολοκλήρωση ενός λειτουργικού συστήματος. Παράλληλα, η διαδικασία ενίσχυσε την εμπειρία σε σύγχρονες τεχνολογίες και μεθοδολογίες, προσφέροντας ένα ολοκληρωμένο τεχνικό και εκπαιδευτικό αποτέλεσμα.

Περίληψη

Η παρούσα διπλωματική εργασία παρουσιάζει τον σχεδιασμό και την υλοποίηση μιας πλατφόρμας διαχείρισης γυμναστηρίου. Στόχος του συστήματος είναι η ψηφιοποίηση και η διευκόλυνση βασικών λειτουργιών, όπως η διαχείριση μελών, συνδρομών και πληρωμών, η οργάνωση μαθημάτων και κρατήσεων, καθώς και η παροχή στοχευμένων προτάσεων προς τους χρήστες. Η υλοποίηση βασίζεται στο πλαίσιο Laravel, αξιοποιεί το Filament για την ανάπτυξη ενός λειτουργικού διαχειριστικού περιβάλλοντος, ενώ για τις πληρωμές χρησιμοποιείται το Stripe Billing. Η διεπαφή σχεδιάστηκε με χρήση Tailwind CSS , ενώ η ανάπτυξη και τοπική εκτέλεση διευκολύνθηκαν μέσω του Laravel Herd.

Στο κείμενο παρουσιάζονται οι βασικές ενότητες της εφαρμογής, τα μοντέλα δεδομένων και οι ροές λειτουργίας, καθώς και ο μηχανισμός συστάσεων που αξιοποιεί ιστορικά δεδομένα αλληλεπίδρασης (π.χ. εμφανίσεις και κλικ) για την ενίσχυση της εμπειρίας χρήστη. Γίνεται επίσης σύντομη αναφορά στον διαχωρισμό δικαιωμάτων μεταξύ ρόλων, ώστε οι χρήστες να έχουν πρόσβαση μόνο στις απαραίτητες λειτουργιές

Το σύστημα παρουσιάζει μια ολοκληρωμένη λύση για την κεντρική διαχείριση των βασικών λειτουργιών ενός γυμναστηρίου, περιορίζει τις χειροκίνητες διαδικασίες και ενισχύει την εξατομίκευση των υπηρεσιών προς τα μέλη.

«Development of a web application for managing gym customers»

(στην αγγλική γλώσσα)

«Dimou Dimitrios»

(στην αγγλική γλώσσα)

Abstract

This thesis presents the design and implementation of a gym management platform. The system aims to digitize and simplify core operations such as member, subscription, and payment management, class scheduling and bookings, as well as providing targeted recommendations to users. The implementation is based on the Laravel framework, uses Filament to build a functional administrative interface, and integrates Stripe Billing for payments. The interface was designed with Tailwind CSS, while development and local execution were facilitated through Laravel Herd.

The text presents the main modules of the application, the data models and operational flows, and the recommendation mechanism that leverages historical interaction data (e.g., impressions and clicks) to enhance the user experience. A brief reference is also made to role-based access control, ensuring users only access the necessary features.

The system provides a comprehensive solution for centrally managing a gym's key operations, reduces manual processes, and strengthens the personalization of services offered to members.

Ευχαριστίες

Θα ήθελα αρχικά να ευχαριστήσω τον επιβλέποντα καθηγητή μου, την οικογένεια μου και τους φίλους μου για την πολύτιμη βοήθεια τους.

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1ο: Εισαγωγή

1.1 Εισαγωγή

Η σύγχρονη λειτουργία ενός γυμναστηρίου απαιτεί τη διαχείριση μεγάλου όγκου πληροφοριών και διαδικασιών, όπως η οργάνωση μελών, συνδρομών, πληρωμών, κρατήσεων και προγραμμάτων μαθημάτων. Σε πολλές περιπτώσεις, οι διαδικασίες αυτές εκτελούνται αποσπασματικά, με χειροκίνητα βήματα ή μέσω ασύνδετων εργαλείων, γεγονός που αυξάνει τα λειτουργικά σφάλματα, μειώνει την αποδοτικότητα και δυσχεραίνει την παροχή προσωποποιημένων υπηρεσιών. Παράλληλα, η ανάγκη για έγκαιρη ενημέρωση, ιστορικότητα δεδομένων και σωστή εξυπηρέτηση των μελών καθιστά απαραίτητη μια ενοποιημένη και αξιόπιστη ψηφιακή λύση.

Στο πλαίσιο αυτό, η παρούσα διπλωματική εργασία επικεντρώνεται στη σχεδίαση και υλοποίηση μιας ολοκληρωμένης πλατφόρμας διαχείρισης γυμναστηρίου, η οποία επιτρέπει την κεντρική οργάνωση των βασικών λειτουργιών και την υποστήριξη τόσο των διαχειριστών όσο και των τελικών χρηστών. Η πλατφόρμα συνδυάζει διαχείριση δεδομένων, αυτοματοποίηση διαδικασιών και παροχή εξατομικευμένων προτάσεων, με στόχο τη βελτίωση της λειτουργικής αποτελεσματικότητας και της εμπειρίας χρήσης. Επιπλέον, η ύπαρξη ενός ενιαίου συστήματος συμβάλλει στη συνέπεια των πληροφοριών, στη μείωση του χρόνου εξυπηρέτησης και στη δυνατότητα εξαγωγής χρήσιμων συμπερασμάτων για τη λειτουργία του οργανισμού.

Ιδιαίτερη έμφαση δίνεται επίσης στη σαφή οργάνωση των δεδομένων και στην ορθότητα των ροών, ώστε η πλατφόρμα να παραμένει αξιόπιστη στην καθημερινή χρήση. Οι διαχειριστές αποκτούν καλύτερη εικόνα της λειτουργίας του γυμναστηρίου, ενώ οι τελικοί χρήστες επωφελούνται από ένα πιο άμεσο και συνεπές περιβάλλον αλληλεπίδρασης. Η αξιοποίηση μηχανισμών συστάσεων προσθέτει ένα στοιχείο εξατομίκευσης, το οποίο μπορεί να ενισχύσει τη δέσμευση και την ικανοποίηση των μελών.

Η επιλογή των τεχνολογιών έγινε με γνώμονα τόσο τις τεχνολογίες που έχουν διδαχθεί στα προηγούμενα ακαδημαϊκά έτη όσο και την αξιοποίηση σύγχρονων εργαλείων που θεωρούνται κατάλληλα για ανάπτυξη ασφαλών, επεκτάσιμων και συντηρήσιμων εφαρμογών. Με αυτόν τον τρόπο επιδιώκεται η σύνδεση της θεωρητικής γνώσης με την πρακτική εφαρμογή σε ένα ρεαλιστικό πληροφοριακό σύστημα, καθώς και η ανάπτυξη μιας λύσης που μπορεί να εξελιχθεί και να καλύψει μελλοντικές ανάγκες.

1.2 Σκοπός και στόχοι

Σκοπός της παρούσας διπλωματικής είναι η ανάλυση, σχεδίαση και υλοποίηση μιας ολοκληρωμένης διαδικτυακής πλατφόρμας για τη διαχείριση γυμναστηρίου, η οποία να καλύπτει τόσο τις εσωτερικές λειτουργίες (διοίκηση, κρατήσεις, πληρωμές) όσο και την εμπειρία του τελικού χρήστη. Η εργασία επιδιώκει να αξιοποιήσει σύγχρονες τεχνολογίες που διδάχθηκαν στο πρόγραμμα σπουδών, συνδυάζοντάς τες με πρακτικά εργαλεία που χρησιμοποιούνται στην αγορά, ώστε το αποτέλεσμα να είναι ρεαλιστικό και άμεσα εφαρμόσιμο. Επιπλέον, στοχεύει να απαντήσει σε μια πραγματική ανάγκη της τοπικής αγοράς, όπου πολλά μικρά γυμναστήρια δεν διαθέτουν σύγχρονες ψηφιακές υπηρεσίες.

Οι βασικοί στόχοι συνοψίζονται ως εξής:

- Ανάπτυξη λειτουργικού συστήματος διαχείρισης συνδρομών, κρατήσεων και πληρωμών.

- Ενσωμάτωση ρόλων χρηστών με διαφορετικά επίπεδα πρόσβασης.
- Υλοποίηση ασφαλών online πληρωμών και τιμολόγησης.
- Παροχή φιλικής εμπειρίας χρήσης για μέλη και διαχειριστές.
- Διερεύνηση μηχανισμού προτάσεων για βελτίωση της εμπλοκής των χρηστών.

1.3 Αντικείμενο και πεδίο εφαρμογής

Αντικείμενο της διπλωματικής είναι η ανάπτυξη μιας web εφαρμογής για τη λειτουργική και διοικητική υποστήριξη ενός γυμναστηρίου. Η εφαρμογή καλύπτει βασικές λειτουργίες όπως διαχείριση μελών και συνδρομών, προγραμματισμό και κρατήσεις μαθημάτων, καταγραφή πληρωμών και έκδοση τιμολογίων, καθώς και διακριτούς ρόλους πρόσβασης για διαχειριστές, προσωπικό και μέλη.

Το πεδίο εφαρμογής περιορίζεται σε ένα μικρό/μεσαίο γυμναστήριο με ανάγκη για ενιαίο ψηφιακό σύστημα. Η πλατφόρμα σχεδιάζεται ώστε να υποστηρίζει καθημερινές λειτουργίες, χωρίς να επεκτείνεται σε εξειδικευμένες υπηρεσίες τρίτων (π.χ. πλήρη λογιστική διαχείριση ή εξωτερικά CRM). Παράλληλα, δίνεται έμφαση στην εμπειρία του τελικού χρήστη και στην προοπτική μελλοντικής επέκτασης.

1.4 Μεθοδολογία και περιορισμοί

Η μεθοδολογία που ακολουθήθηκε περιλαμβάνει: (α) ανάλυση απαιτήσεων και καταγραφή ροών λειτουργίας ενός γυμναστηρίου, (β) σχεδίαση της αρχιτεκτονικής, των δεδομένων και των βασικών διεπαφών, (γ) υλοποίηση της εφαρμογής με σύγχρονες τεχνολογίες web (Laravel, Filament, Tailwind CSS) και αξιοποίηση υπηρεσιών πληρωμών (Stripe), (δ) έλεγχο λειτουργικότητας με δοκιμές σε βασικά σενάρια χρήσης και (ε) τεκμηρίωση των αποτελεσμάτων. Η προσέγγιση ήταν σταδιακή και επαναληπτική, ώστε να υπάρχει συνεχής ανατροφοδότηση και βελτίωση.

Οι περιορισμοί της εργασίας σχετίζονται με το διαθέσιμο χρονικό πλαίσιο και το εύρος υλοποίησης. Η εφαρμογή επικεντρώνεται σε μικρές/μεσαίες δομές και δεν καλύπτει εξειδικευμένες επιχειρησιακές ανάγκες (π.χ. πλήρη λογιστική διαχείριση, πολυκαταστηματική λειτουργία ή προχωρημένα εργαλεία CRM). Επιπλέον, δεν υπήρχε η δυνατότητα για να λειτουργήσει το σύστημα σε κανονικές συνθήκες χρήσης με πραγματικούς χρήστες και συνεχή ροή δεδομένων, γεγονός που περιορίζει την αξιολόγηση των αλγορίθμων προτάσεων σε πραγματικό χρόνο.

1.5 Δομή εργασίας

Η εργασία οργανώνεται ως εξής: Στο Κεφάλαιο 1 παρουσιάζονται η εισαγωγή, ο σκοπός, το αντικείμενο και η μεθοδολογία της διπλωματικής. Στο Κεφάλαιο 2 γίνεται συνοπτική θεωρητική ανασκόπηση των τεχνολογιών και βασικών εννοιών που στηρίζουν την πλατφόρμα. Στο Κεφάλαιο 3 περιγράφονται οι απαιτήσεις του συστήματος και ο σχεδιασμός των δεδομένων και των βασικών λειτουργιών. Στο Κεφάλαιο 4 αναλύεται η υλοποίηση, η αρχιτεκτονική και τα κύρια υποσυστήματα της εφαρμογής. Τέλος, στο Κεφάλαιο 5 παρουσιάζονται τα αποτελέσματα, η αξιολόγηση της λύσης, τα συμπεράσματα και σύντομες προτάσεις για μελλοντικές επεκτάσεις.

Κεφάλαιο 2ο: Τεχνολογίες και Εργαλεία

2.1 Εισαγωγή και ορολογία

Στο παρόν κεφάλαιο παρουσιάζονται οι τεχνολογίες και τα εργαλεία που αξιοποιήθηκαν για την ανάπτυξη της εφαρμογής διαχείρισης γυμναστηρίου. Στόχος είναι να τεκμηριωθεί γιατί επιλέχθηκε κάθε τεχνολογία και πώς συμβάλλει στη συνολική λειτουργία του συστήματος, τόσο σε επίπεδο υποδομής όσο και σε επίπεδο εμπειρίας χρήστη. Η επιλογή δεν έγινε αποκλειστικά με βάση την ευκολία υλοποίησης, αλλά και με γνώμονα την αξιοπιστία, τη συντηρησιμότητα και την ευκολία επέκτασης στο μέλλον.

Οι επόμενες ενότητες καλύπτουν το βασικό πλαίσιο ανάπτυξης της εφαρμογής, το περιβάλλον διαχείρισης (admin panel), τη διαχείριση πληρωμών και συνδρομών, το εργαλείο για το authentication, το styling της διεπαφής, καθώς και τα εργαλεία για το bundling και το τοπικό περιβάλλον ανάπτυξης. Τέλος, παρουσιάζονται τα εργαλεία παραγωγικότητας που χρησιμοποιήθηκαν κατά την ανάπτυξη, με στόχο να αποτυπωθεί συνολικά το τεχνολογικό οικοσύστημα της εργασίας.

Πριν την παρουσίαση των τεχνολογιών, ορίζονται βασικοί όροι που θα χρησιμοποιούνται σε όλο το κείμενο. Η εφαρμογή βασίζεται στο πρότυπο MVC (Model–View–Controller), όπου το Model διαχειρίζεται τα δεδομένα, το View την προβολή και το Controller τη ροή/λογική. Οι διαδρομές (routes) αντιστοιχούν URL σε λειτουργίες και καθορίζουν την πλοήγηση μέσα στον ιστότοπο (site). Στη συνέχεια χρησιμοποιούνται σταθερά οι όροι: σύνδεση (login), εγγραφή (register), πίνακας ελέγχου χρήστη (dashboard), πίνακας διαχείρισης (admin panel), πλάνα συνδρομής (plans) και προπονήσεις (classes).

2.2 Laravel

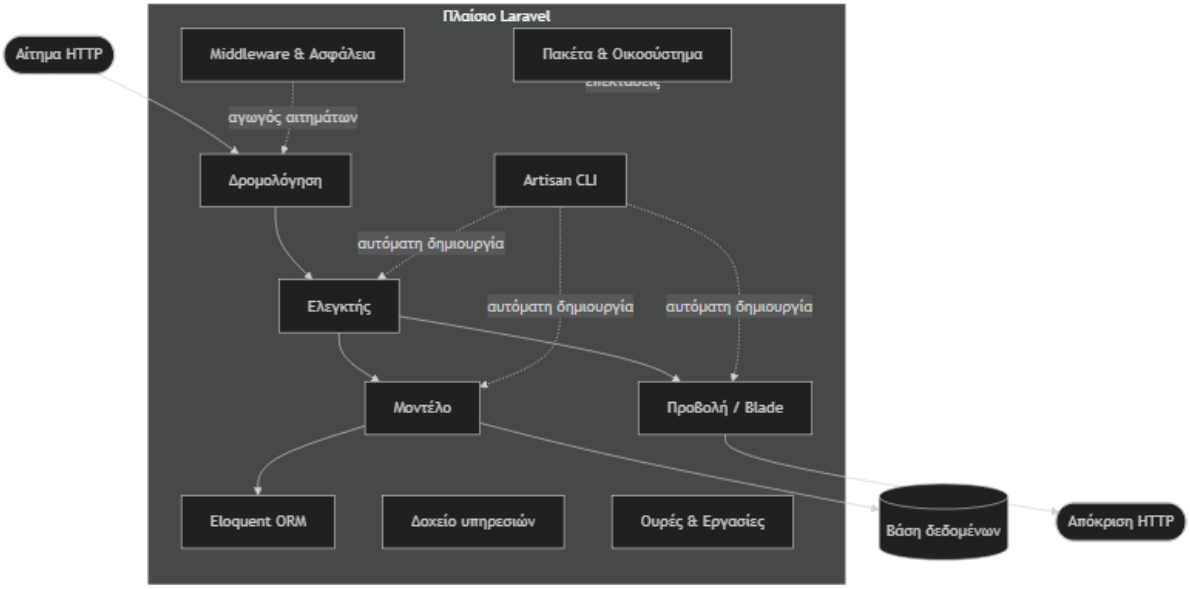
Το Laravel είναι ένα σύγχρονο PHP framework που προσφέρει ένα ολοκληρωμένο περιβάλλον ανάπτυξης για web εφαρμογές. Χαρακτηρίζεται από καθαρή αρχιτεκτονική, έμφαση στην παραγωγικότητα και μεγάλη κοινότητα υποστήριξης. Βασίζεται στο πρότυπο MVC (Model–View–Controller), το οποίο χωρίζει την εφαρμογή σε τρία επίπεδα:

- Model: περιγράφει τα δεδομένα και τη λογική πρόσβασης στη βάση (π.χ. χρήστες, συνδρομές, κρατήσεις).
- View: αφορά την παρουσίαση, δηλαδή τα templates που βλέπει ο χρήστης.
- Controller: λειτουργεί ως “συντονιστής”, λαμβάνει αιτήματα, χειρίζεται τη λογική και επιστρέφει την κατάλληλη απάντηση/θέαση.

Αυτός ο διαχωρισμός κάνει τον κώδικα πιο οργανωμένο, ευκολότερο στη συντήρηση και πιο επεκτάσιμο.

Στο Σχήμα 2.1 παρουσιάζεται συνοπτικά η ροή MVC και τα βασικά δομικά στοιχεία του Laravel, ώστε να γίνει πιο σαφής η θέση τους στην αρχιτεκτονική της εφαρμογής

Ένα από τα βασικά πλεονεκτήματα του Laravel είναι το πλούσιο σύνολο εργαλείων που προσφέρει “out of the box”. Το Artisan CLI επιτρέπει την αυτοματοποίηση κοινών εργασιών (δημιουργία controllers, migrations, seeders, jobs κ.ά.), ενώ το Eloquent ORM παρέχει έναν εύχρηστο τρόπο πρόσβασης και διαχείρισης δεδομένων μέσω μοντέλων, χωρίς άμεση χρήση SQL. Επιπλέον, διαθέτει έτοιμες λύσεις για routing, middleware, authentication/authorization, notifications, queues και caching, μειώνοντας σημαντικά τον χρόνο ανάπτυξης.



Σχήμα 2.1:Συνοπτική απεικόνιση MVC και βασικών στοιχείων του Laravel (ιδία απεικόνιση, βασισμένη στην τεκμηρίωση του Laravel

Στην παρούσα διπλωματική, το Laravel αποτέλεσε τη βάση για την υλοποίηση των βασικών λειτουργιών της πλατφόρμας, όπως η διαχείριση χρηστών, κρατήσεων, συνδρομών και πληρωμών. Η σταθερότητα, η οργάνωση και το οικοσύστημα πακέτων του συνέβαλαν σε μια πιο ολοκληρωμένη και αξιόπιστη υλοποίηση, δίνοντας τη δυνατότητα να επικεντρωθεί η εργασία στη λειτουργικότητα και όχι στην επανεφεύρεση βασικών μηχανισμών.

2.3 Laravel starter kits(Laravel Breeze) και Filament (admin panel) .

Στο πλαίσιο της αρχικής υλοποίησης, χρειαζόταν πρώτα ένα σταθερό και ελαφρύ σημείο εκκίνησης για την αυθεντικοποίηση. Γι' αυτό επιλέχθηκε το Laravel Breeze. Κυρίως, το Breeze είναι ένα ελάχιστο starter kit αυθεντικοποίησης: δημιουργεί άμεσα τις βασικές ροές εισόδου και εγγραφής (login, register, reset κωδικού, επαλήθευση email, επιβεβαίωση κωδικού) και εγκαθιστά το βασικό UI με Blade/Tailwind. Ταυτόχρονα, δημοσιεύει (publish) τον πραγματικό κώδικα των routes, controllers και views μέσα στο project, ώστε να είναι πλήρως παραμετροποιήσιμο και να μην υπάρχει "κρυφή" λογική. Έτσι, λειτουργεί ως καθαρό σημείο εκκίνησης: δίνει έτοιμη, ασφαλή αυθεντικοποίηση, αλλά αφήνει στον προγραμματιστή τον πλήρη έλεγχο της εξέλιξης της εφαρμογής.

Καθώς όμως η εφαρμογή απαιτούσε πιο ολοκληρωμένο περιβάλλον διαχείρισης (users, ρόλοι, CRUD λειτουργίες, dashboards), το Breeze αντικαταστάθηκε από το Filament. Το Filament παρέχει έτοιμο admin panel πάνω στο Laravel, με μηχανισμούς για resources, φόρμες, πίνακες, φίλτρα, actions και πολιτικές πρόσβασης. Έτσι, η διαχείριση της πλατφόρμας συγκεντρώθηκε σε ένα ενιαίο περιβάλλον, βελτιώνοντας την παραγωγικότητα και μειώνοντας την ανάγκη για ξεχωριστές custom διεπαφές.

Η μετάβαση αυτή επέτρεψε καλύτερη οργάνωση των λειτουργιών διαχείρισης, πιο σταθερή εμπειρία για τους διαχειριστές και ευκολότερη επέκταση του συστήματος, χωρίς να χαθεί η ασφάλεια και η συμβατότητα με τον πυρήνα του Laravel..

2.4 Laravel cashier / Stripe (συνδρομές και πληρωμές)

Για τη διαχείριση συνδρομών και online πληρωμών επιλέχθηκε ο συνδυασμός του Laravel Cashier με το Stripe. Το Stripe είναι μια ευρέως χρησιμοποιούμενη πλατφόρμα πληρωμών που παρέχει ασφαλείς

συναλλαγές, υποστήριξη καρτών, εργαλεία τιμολόγησης και μηχανισμούς ενημέρωσης μέσω webhooks. Προσφέρει τόσο υποδομή για απλές χρεώσεις όσο και για επαναλαμβανόμενες πληρωμές (subscriptions), γεγονός που ταιριάζει άμεσα στο μοντέλο συνδρομών ενός γυμναστηρίου.

Το Laravel Cashier αποτελεί το επίσημο billing πακέτο του Laravel και λειτουργεί ως ενδιάμεσο επίπεδο ανάμεσα στην εφαρμογή και το Stripe. Παρέχει έτοιμες κλάσεις και μεθόδους για τη δημιουργία πελατών, τη διαχείριση πλάνων, την ενεργοποίηση/παύση συνδρομών, καθώς και την παρακολούθηση της κατάστασης πληρωμών χωρίς να απαιτούνται χαμηλού επιπέδου API κλήσεις. Έτσι, μειώνεται η πολυπλοκότητα του κώδικα και γίνεται πιο ασφαλής η ενσωμάτωση πληρωμών.

Ιδιαίτερα σημαντικός είναι ο ρόλος των webhooks, μέσω των οποίων το Stripe ενημερώνει την εφαρμογή για γεγονότα όπως επιτυχής πληρωμή, αποτυχημένη χρέωση ή ακύρωση συνδρομής. Με αυτόν τον τρόπο το σύστημα ενημερώνεται αυτόματα και σε πραγματικό χρόνο, εξασφαλίζοντας συνέπεια ανάμεσα στα δεδομένα της εφαρμογής και τις συναλλαγές στο Stripe. Επιπλέον, το Stripe παρέχει test mode, επιτρέποντας την ασφαλή δοκιμή πληρωμών χωρίς πραγματικές χρεώσεις, κάτι που είναι απαραίτητο στη φάση ανάπτυξης και ελέγχου. Στο Σχήμα 2.2 αποτυπώνεται συνοπτικά η ροή πληρωμών και ο ρόλος των webhooks στη συγχρονισμένη ενημέρωση της εφαρμογής.

Σχήμα 2.2: Ροή συνδρομής και πληρωμής με Stripe και Laravel Cashier (ιδία απεικόνιση)

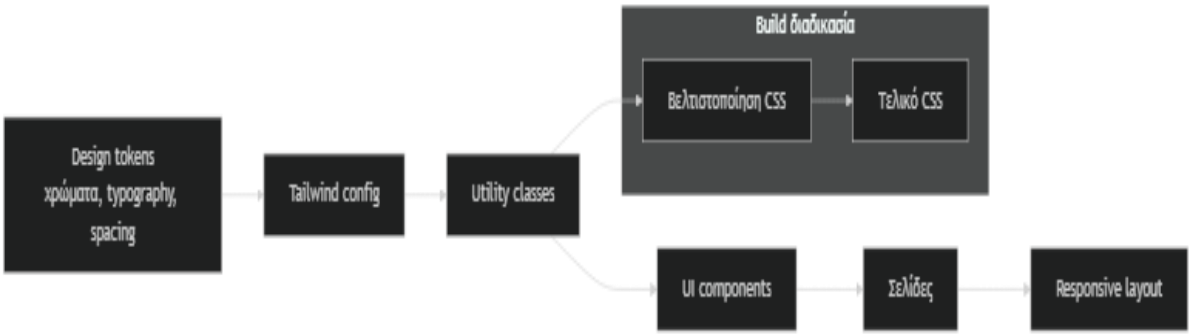
Η επιλογή Cashier/Stripe προσέφερε μια αξιόπιστη και επεκτάσιμη βάση για οικονομικές συναλλαγές, καλύπτοντας τις ανάγκες της πλατφόρμας χωρίς να απαιτείται ανάπτυξη από το μηδέν. Με αυτόν τον τρόπο, η εφαρμογή αποκτά επαγγελματικό επίπεδο στην υποστήριξη συνδρομών και πληρωμών, με έμφαση στην ασφάλεια και τη διαχειρισιμότητα..

2.5 Tailwind CSS

Για τη σχεδίαση και το styling της διεπαφής χρησιμοποιήθηκε το Tailwind CSS, ένα utility-first framework που βασίζεται σε μικρές κλάσεις (utility classes) οι οποίες εφαρμόζονται απευθείας στο HTML. Αντί να γράφεται εκτενές custom CSS, ο σχεδιασμός προκύπτει από συνδυασμούς κλάσεων για spacing, typography, χρώματα, layout και responsive behavior. Αυτό επιτρέπει γρήγορες αλλαγές, συνεπή αισθητική και μικρότερο overhead συντήρησης.

Η προσέγγιση του Tailwind βοηθά ιδιαίτερα σε έργα όπου η διεπαφή εξελίσσεται σταδιακά, καθώς μειώνει το χρόνο πειραματισμού και κάνει πιο άμεση την προσαρμογή του UI. Παράλληλα, μέσω του configuration μπορεί να οριστεί ένα “design system” (π.χ. χρωματική παλέτα, μεγέθη γραμματοσειρών, breakpoints), ώστε η εφαρμογή να διατηρεί ομοιομορφία σε όλες τις σελίδες. Στην παρούσα εργασία, το Tailwind χρησιμοποιήθηκε για την εμφάνιση του δημόσιου μέρους της εφαρμογής αλλά και για την εναρμόνιση επιμέρους στοιχείων της διεπαφής με το συνολικό ύφος.

Το αποτέλεσμα είναι μια καθαρή και responsive διεπαφή, με συνεπή αισθητική και ευκολία επέκτασης. Επιπλέον, η σύνδεσή του με το build σύστημα (Vite) επιτρέπει τη βελτιστοποίηση του τελικού CSS, ώστε να παραμένει ελαφρύ και αποδοτικό.



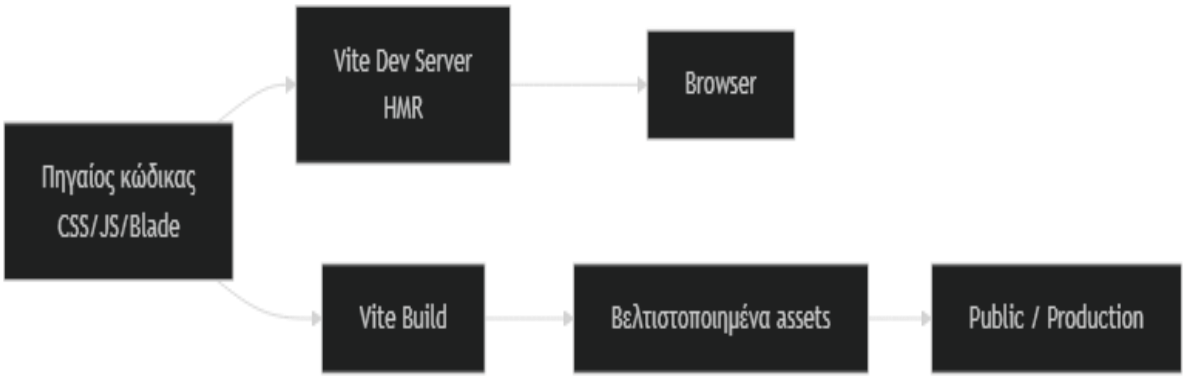
Σχήμα 2.3: Ροή σχεδιασμού και παραγωγής CSS με Tailwind

2.6 Vite (bundling & dev server)

Για το bundling των assets και την υποστήριξη του τοπικού development χρησιμοποιήθηκε το Vite, ένα σύγχρονο εργαλείο build που δίνει έμφαση στην ταχύτητα και στην άμεση εμπειρία ανάπτυξης. Σε αντίθεση με παλαιότερα εργαλεία bundling, το Vite αξιοποιεί ES modules και προσφέρει γρήγορο dev server με hot module replacement (HMR), επιτρέποντας άμεση ανανέωση αλλαγών στο frontend χωρίς πλήρες reload της σελίδας.

Στην παρούσα εργασία, το Vite συνέβαλε στη γρήγορη ανάπτυξη και δοκιμή των διεπαφών, καθώς επιτρέπει την άμεση προεπισκόπηση αλλαγών σε CSS και JavaScript. Παράλληλα, κατά το build δημιουργεί βελτιστοποιημένα αρχεία (minified CSS/JS), μειώνοντας το μέγεθος των assets και βελτιώνοντας την απόδοση στο production περιβάλλον. Η ενσωμάτωσή του με το Laravel είναι πλέον στάνταρ επιλογή, γεγονός που απλοποιεί τη ρύθμιση του build pipeline και εξασφαλίζει συνεπή τρόπο διαχείρισης των static πόρων.

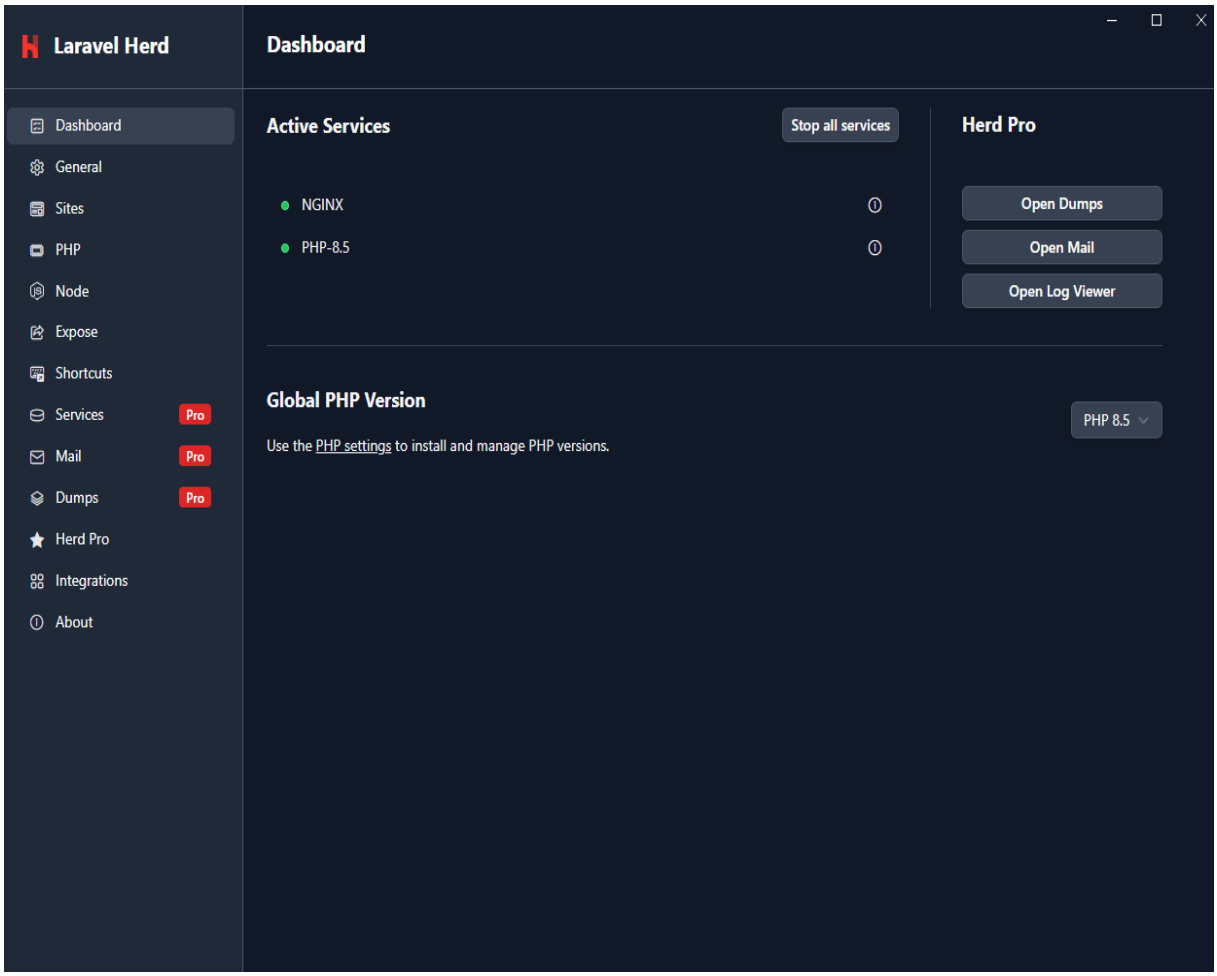
Επιπλέον, το Vite επιτρέπει καλύτερη οργάνωση του frontend κώδικα και καθιστά πιο εύκολη τη συντήρηση καθώς μεγαλώνει η εφαρμογή. Με αυτόν τον τρόπο, υποστηρίζει τόσο την ταχύτητα ανάπτυξης όσο και τη σταθερότητα του τελικού αποτελέσματος.



Σχήμα 2.4: Ροή ανάπτυξης και build με Vite

2.7 Herd

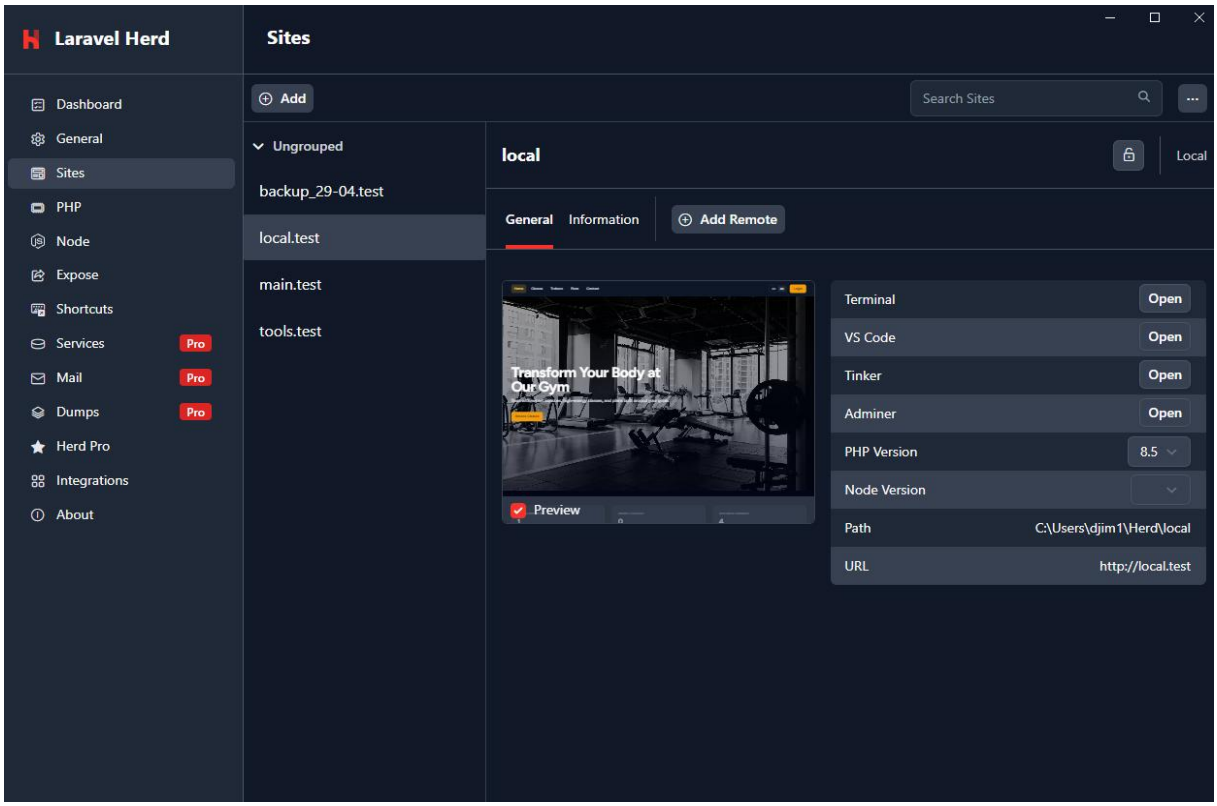
Για το τοπικό περιβάλλον ανάπτυξης χρησιμοποιήθηκε το **Laravel Herd**, ένα ελαφρύ εργαλείο που παρέχει έτοιμο και αξιόπιστο περιβάλλον για την εκτέλεση εφαρμογών Laravel σε Windows/macOS. Το Herd περιλαμβάνει PHP, web server και βασικά εργαλεία ανάπτυξης, προσφέροντας γρήγορη εγκατάσταση και απλοποιημένη διαχείριση πολλαπλών projects, χωρίς πολύπλοκες ρυθμίσεις.



Εικόνα 2.5: Πίνακας ελέγχου του Laravel Herd με ενεργές υπηρεσίες και έκδοση PHP

Στο Σχήμα 2.5 παρουσιάζεται το περιβάλλον του Laravel Herd όπως χρησιμοποιήθηκε στη διπλωματική. Το πρώτο στιγμιότυπο δείχνει τον πίνακα ελέγχου, όπου φαίνονται ενεργές οι βασικές υπηρεσίες (π.χ. Nginx και PHP) και η επιλεγμένη έκδοση PHP.

Η επιλογή του Herd βοήθησε ιδιαίτερα στην ταχεία εκκίνηση της ανάπτυξης, καθώς δεν απαιτήθηκε χειροκίνητη ρύθμιση Apache/Nginx ή ξεχωριστές PHP εκδόσεις. Παράλληλα, προσφέρει ευκολία στην αλλαγή μεταξύ projects, αυτόματο σερβίρισμα τοπικών domains και συμβατότητα με το οικοσύστημα του Laravel, κάτι που μειώνει πιθανά προβλήματα κατά τη διάρκεια της υλοποίησης.



Εικόνα 2.6: Διαχείριση τοπικού site στο Laravel Herd (ρυθμίσεις, path και local domain).

Στο Σχήμα 2.5 παρουσιάζεται η διαχείριση ενός τοπικού site, με πληροφορίες για το project, το path και το αντίστοιχο local domain, επιβεβαιώνοντας τον απλό τρόπο με τον οποίο το Herd οργανώνει και εκκινεί εφαρμογές.

Με αυτόν τον τρόπο, το Herd λειτούργησε ως σταθερή βάση ανάπτυξης, επιτρέποντας γρήγορο testing και ομαλή ροή εργασίας, κάτι που είναι κρίσιμο σε μια διπλωματική όπου ο χρόνος ανάπτυξης είναι περιορισμένος.

2.8 Visual Studio Code (Ide/εργαλεία)

Για την ανάπτυξη της εφαρμογής χρησιμοποιήθηκε το Visual Studio Code (VS Code), ένα ελαφρύ αλλά ισχυρό IDE που προσφέρει υψηλή παραγωγικότητα και μεγάλη επεκτασιμότητα. Το VS Code υποστηρίζει ενσωματωμένο τερματικό, διαχείριση Git, syntax highlighting και δυνατότητες αυτόματης συμπλήρωσης, τα οποία επιταχύνουν την καθημερινή εργασία. Ιδιαίτερα χρήσιμη είναι η δυνατότητα εγκατάστασης επεκτάσεων, όπως για PHP/Laravel, Blade templates και Tailwind CSS, που βελτιώνουν την ακρίβεια και την ταχύτητα συγγραφής κώδικα.

Στο πλαίσιο της διπλωματικής, το VS Code βοήθησε στη γρήγορη πλοήγηση σε μεγάλο codebase, στην οργάνωση αρχείων και στην αποτελεσματική διαχείριση των assets. Μέσω του ενσωματωμένου terminal και της υποστήριξης task runners, έγινε πιο εύκολη η εκτέλεση εντολών (π.χ. artisan, migrations, build scripts), ενώ οι linting/formatting λειτουργίες συνέβαλαν στη διατήρηση σταθερού ύφους στον κώδικα. Έτσι, το VS Code αποτέλεσε βασικό εργαλείο παραγωγικότητας, υποστηρίζοντας όλη τη ροή ανάπτυξης από την υλοποίηση έως τον έλεγχο και τη συντήρηση της εφαρμογής.

2.9 Επίλογος Κεφαλαίου

Στο κεφάλαιο αυτό παρουσιάστηκαν οι βασικές τεχνολογίες και τα εργαλεία που υποστήριξαν την ανάπτυξη της εφαρμογής. Από το Laravel ως κεντρικό framework και το Filament για το διοικητικό περιβάλλον, μέχρι το Cashier/Stripe για τις πληρωμές, το Tailwind για το UI, το Vite για το build και το Herd για το τοπικό περιβάλλον, κάθε επιλογή συνέβαλε στην αξιοπιστία, την ταχύτητα υλοποίησης και τη συντηρησιμότητα του συστήματος. Επιπλέον, η χρήση του VS Code ως βασικού IDE ενίσχυσε την παραγωγικότητα και την οργάνωση της εργασίας.

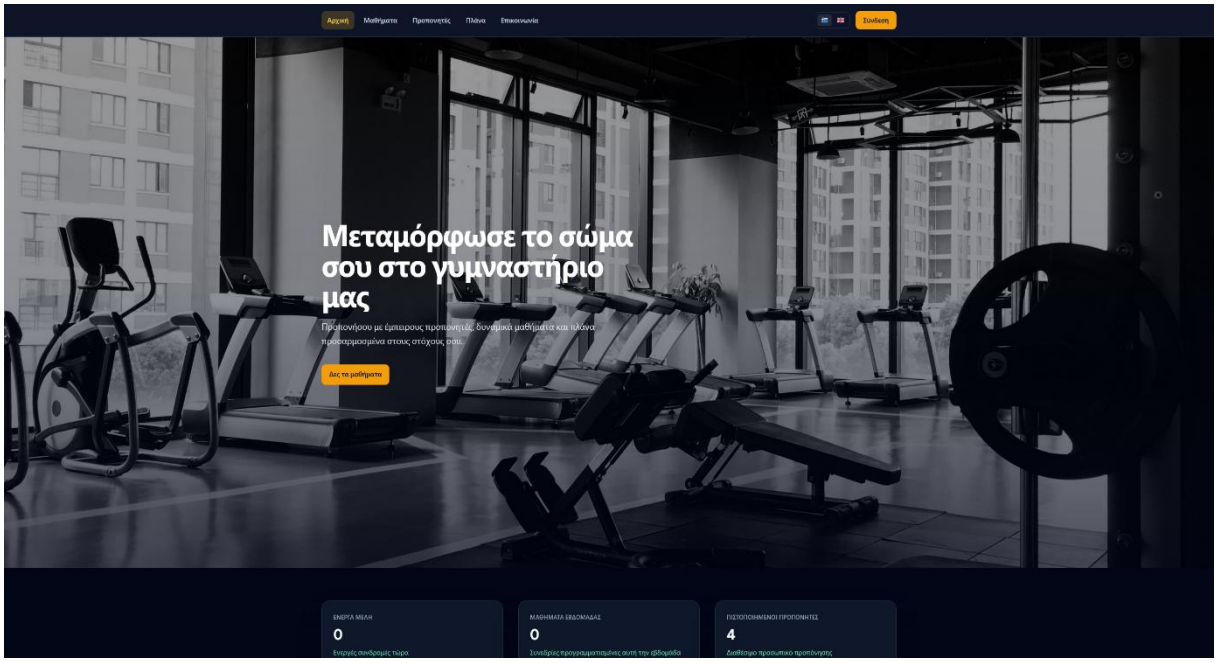
Η σύνθεση αυτών των εργαλείων δημιούργησε ένα ολοκληρωμένο τεχνολογικό οικοσύστημα, το οποίο κάλυψε τόσο τις ανάγκες της ανάπτυξης όσο και τις απαιτήσεις μιας πραγματικής εφαρμογής. Οι επιλογές αυτές δεν έγιναν μόνο με γνώμονα την ευκολία, αλλά και τη σταθερότητα, την ασφάλεια και τη δυνατότητα μελλοντικής επέκτασης. Με αυτόν τον τρόπο, τίθενται οι τεχνολογικές βάσεις για τα επόμενα κεφάλαια, όπου θα παρουσιαστεί η ανάλυση απαιτήσεων, ο σχεδιασμός και η υλοποίηση της πλατφόρμας.

Κεφάλαιο 3ο: Παρουσίαση Διεπαφών Χρήστη

Το Κεφάλαιο 3 παρουσιάζει τις βασικές διεπαφές της εφαρμογής με στιγμιότυπα οθόνης, ώστε να φαίνεται καθαρά η εμπειρία χρήσης σε κάθε ρόλο. Αρχικά παρουσιάζονται οι δημόσιες σελίδες του ιστότοπου (αρχική, προπονήσεις, προπονητές, πλάνα συνδρομής, επικοινωνία), στη συνέχεια οι οθόνες σύνδεσης/εγγραφής και ο πίνακας ελέγχου χρήστη, ενώ κλείνει με τον πίνακα διαχείρισης για τις λειτουργίες του διαχειριστή. Για κάθε οθόνη δίνονται σύντομες περιγραφές και τα βασικά σημεία λειτουργικότητας.

3.1 Δημόσιες σελίδες ιστότοπου: Αρχική, προπονήσεις, Προπονητές, Πλάνα συνδρομής, Επικοινωνία

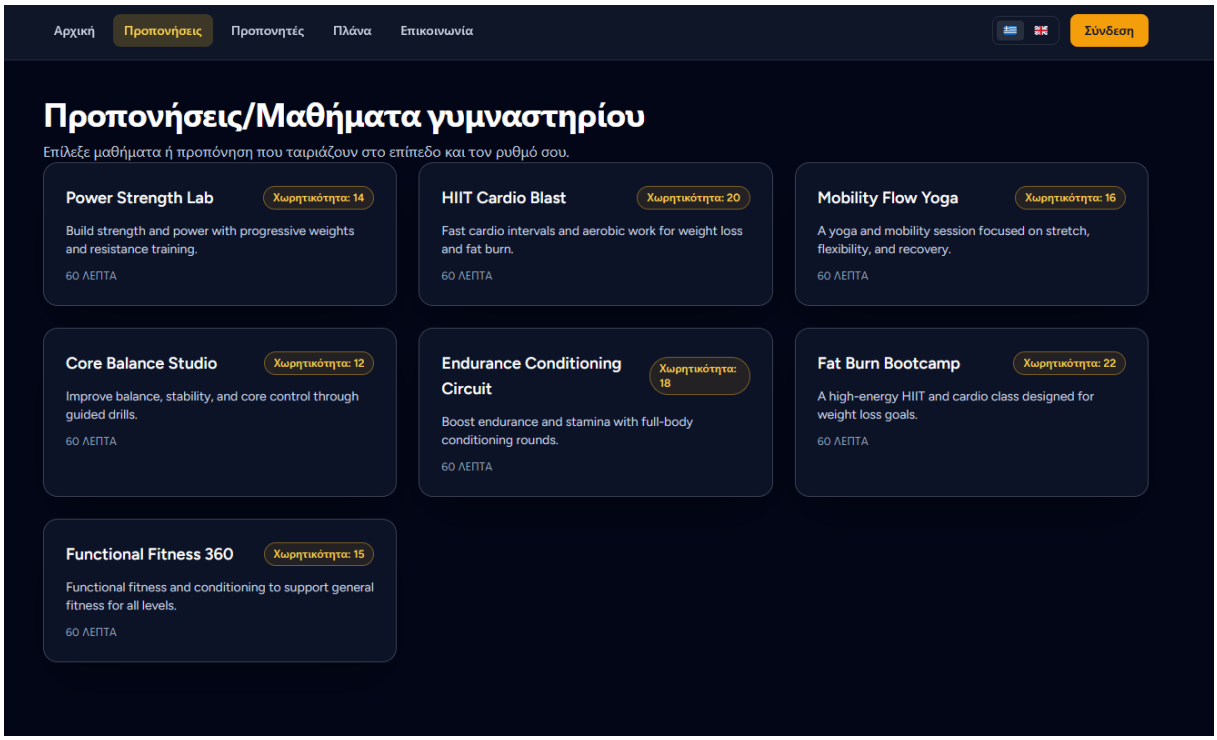
Η αρχική σελίδα λειτουργεί ως σημείο υποδοχής με κεντρικό μήνυμα και άμεση πρόσβαση στις προπονήσεις. Περιλαμβάνει συνοπτικούς δείκτες δραστηριότητας για γρήγορη εικόνα του γυμναστηρίου.



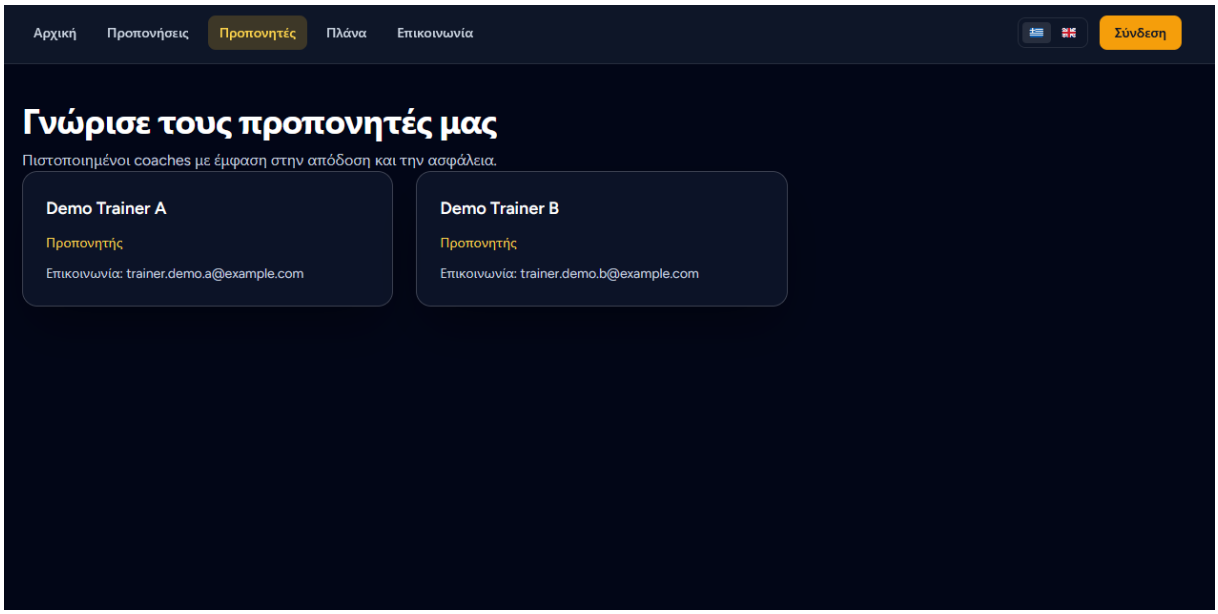
Εικόνα 3.1: Στιγμιότυπο αρχικής σελίδας ιστότοπου. Πηγή hero εικόνας: https://unsplash.com/photos/gym-equipment-inside-room-20jX9b35r_M[8] , διαθέσιμη για χρήση με την άδεια χρήσης του unsplash.com

Η σελίδα μαθημάτων παρουσιάζεται ως λίστα διαθέσιμων μαθημάτων με τίτλο, σύντομη περιγραφή και βασικές πληροφορίες διάρκειας/έντασης, στην εικόνα 3.2. Η διάταξη διευκολύνει τη γρήγορη σύγκριση επιλογών.

Στην εικόνα 3.3, προβάλλονται οι διαθέσιμοι προπονητές με βασικά στοιχεία και στοιχεία επικοινωνίας. Στόχος είναι η άμεση γνωριμία του χρήστη με το ανθρώπινο δυναμικό.

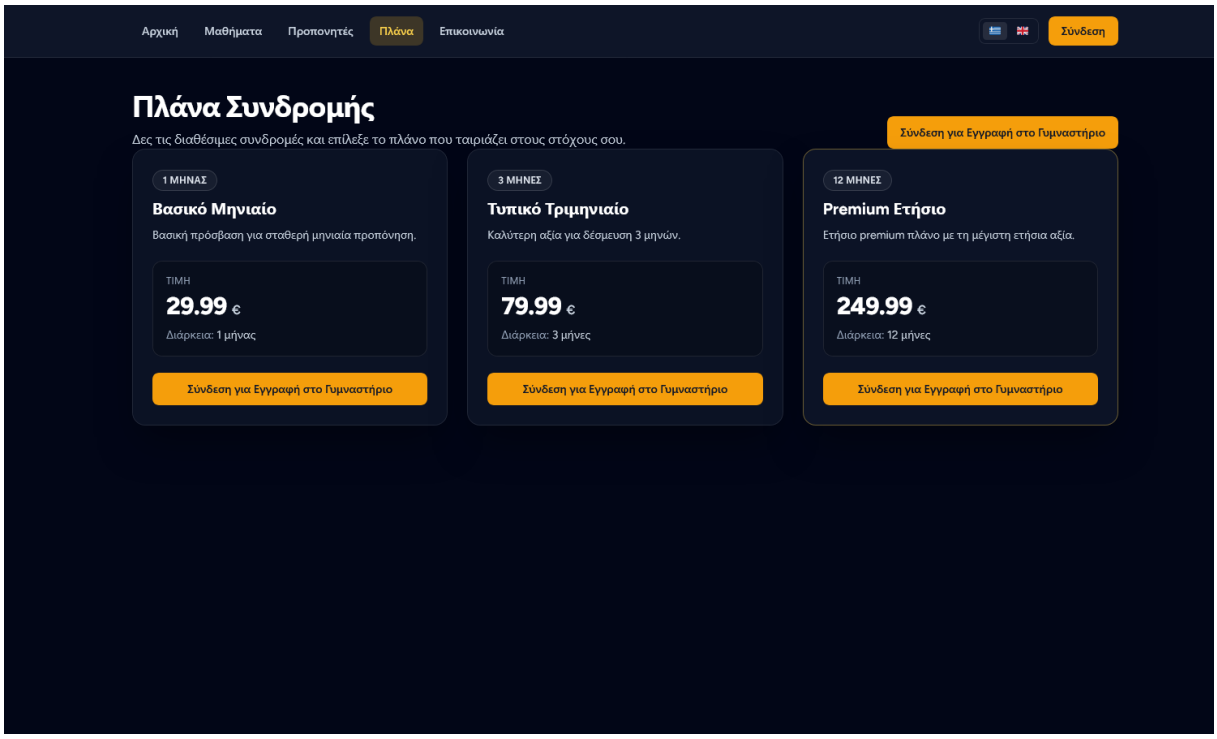


Εικόνα 3.2: Στιγμιότυπο σελίδας ιστότοπου προπονήσεων/μαθημάτων γυμναστηρίου



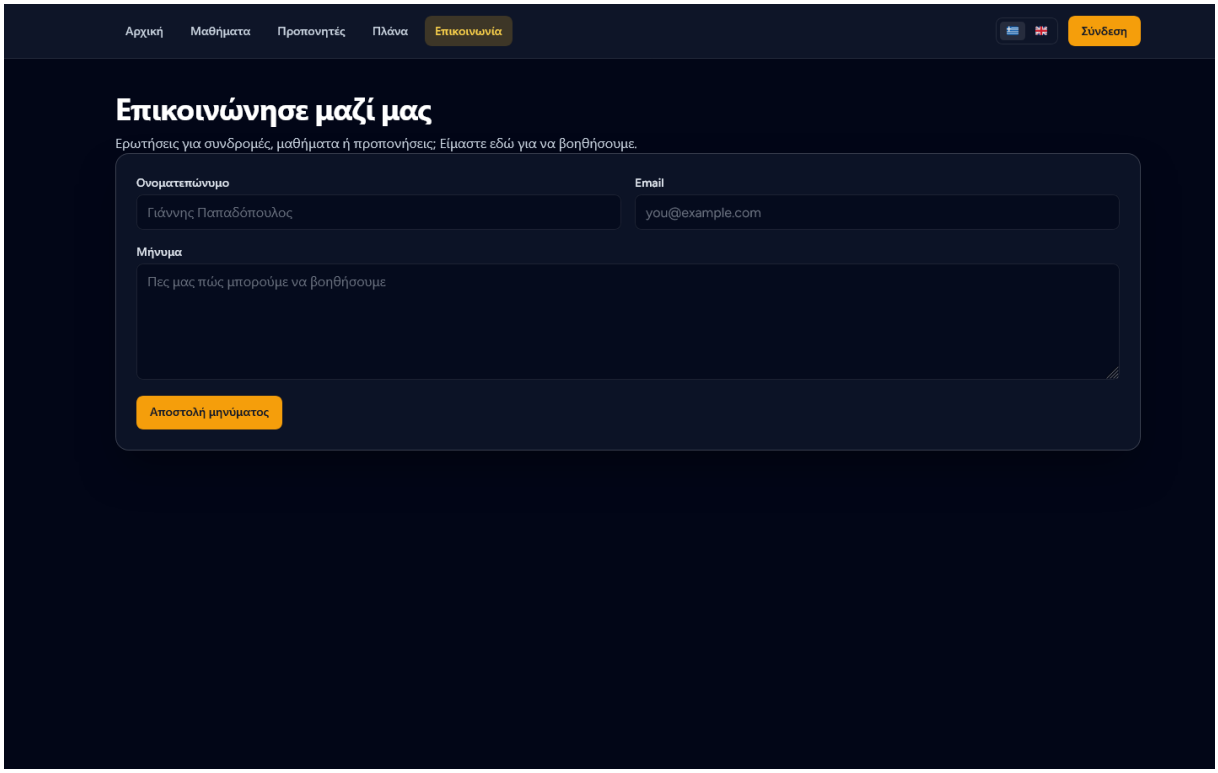
Εικόνα 3.3: Στιγμιότυπο σελίδας ιστότοπου προπονητών γυμναστηρίου

Στην εικόνα 3.4, εμφανίζονται τα διαθέσιμα πλάνα με διάρκεια και τιμή, ώστε ο χρήστης να συγκρίνει εύκολα τις επιλογές. Τα κουμπιά δράσης οδηγούν στη διαδικασία εγγραφής/σύνδεσης.



Εικόνα 3.4: Στιγμιότυπο σελίδας πλάνων συνδρομής

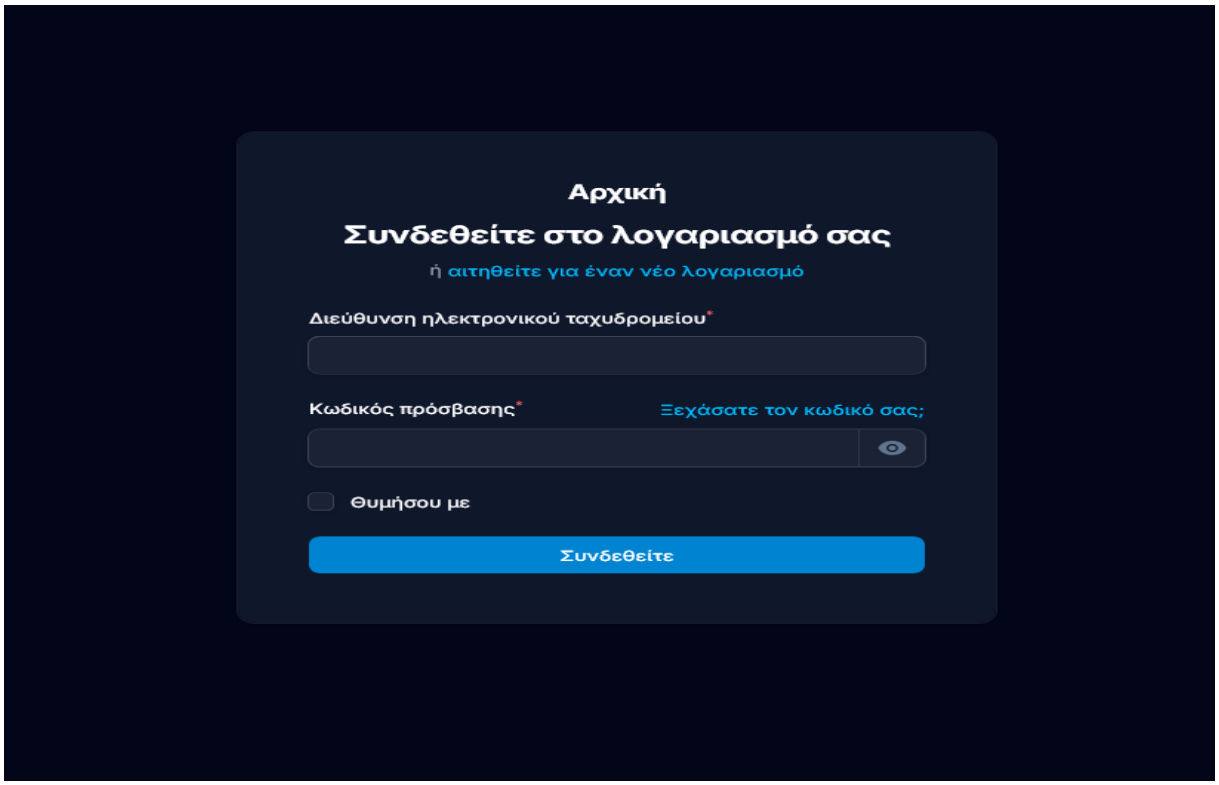
Στην εικόνα 3.5, είναι μια φόρμα επικοινωνίας για ερωτήσεις και αιτήματα, με πεδία ονόματος, email και μηνύματος. Παρέχει έναν απλό τρόπο αποστολής μηνύματος προς τη διαχείριση.



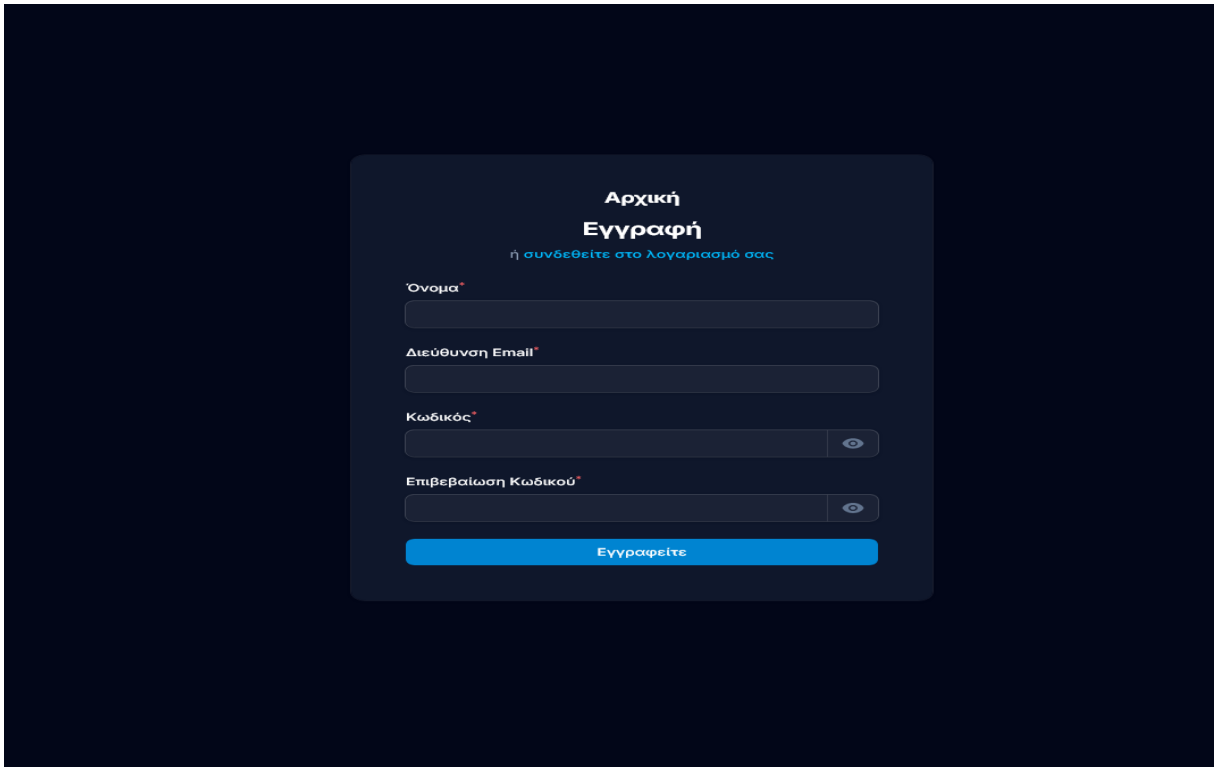
Εικόνα 3.5: Στιγμιότυπο σελίδας φόρμας επικοινωνίας

3.2 Αυθεντικοποίηση (Σύνδεση και εγγραφή)

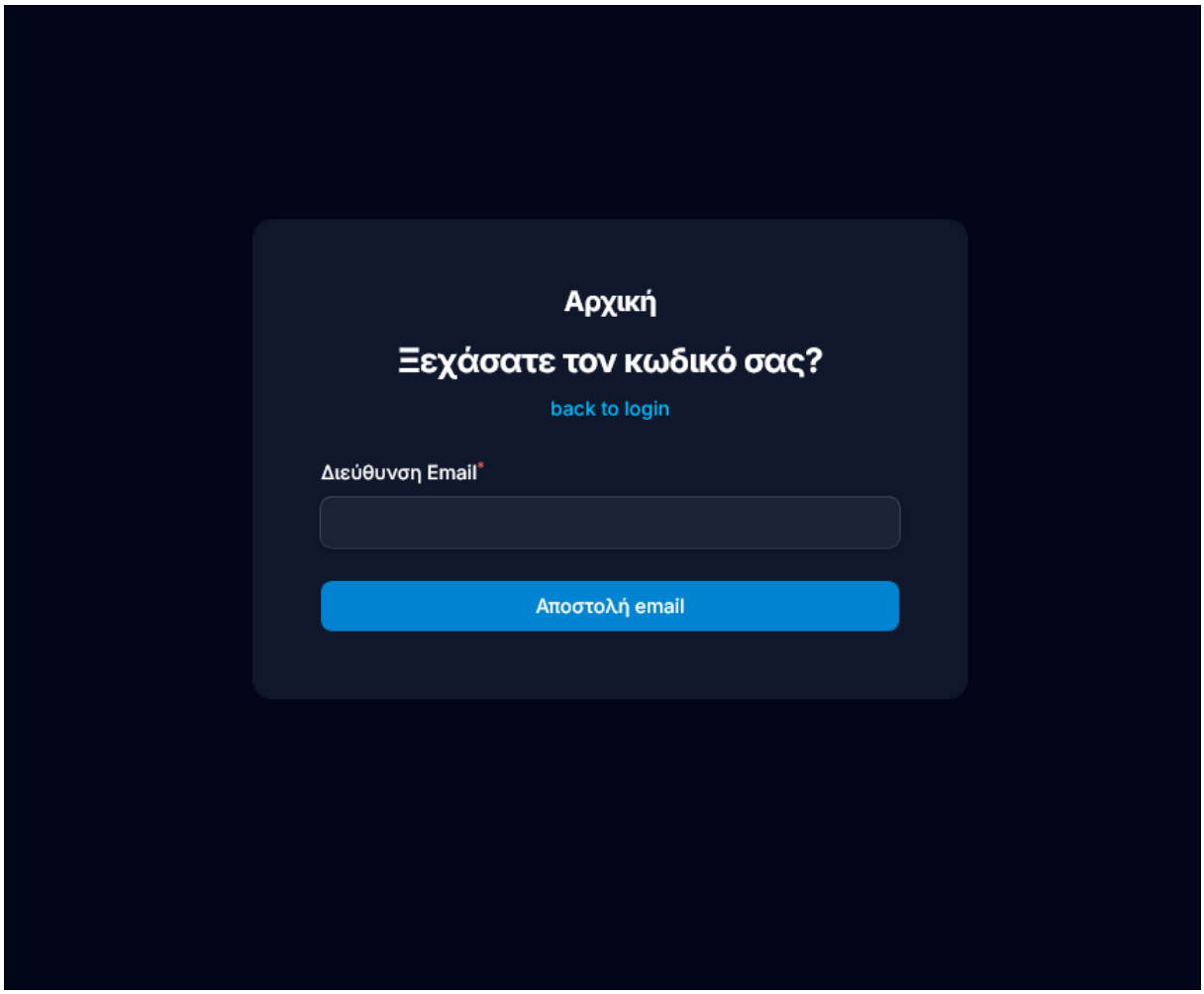
Η ενότητα αυτή παρουσιάζει τις οθόνες αυθεντικοποίησης, οι οποίες επιτρέπουν την ασφαλή πρόσβαση των χρηστών στην πλατφόρμα. Η διεπαφή διατηρεί ενιαία αισθητική με το υπόλοιπο site και προσφέρει σαφή ροή για σύνδεση, δημιουργία λογαριασμού και ανάκτηση κωδικού.



Εικόνα 3.6: Στιγμιότυπο σύνδεσης στο λογαριασμό χρήστη



Εικόνα 3.7: Στιγμιότυπο δημιουργίας λογαριασμού χρήστη

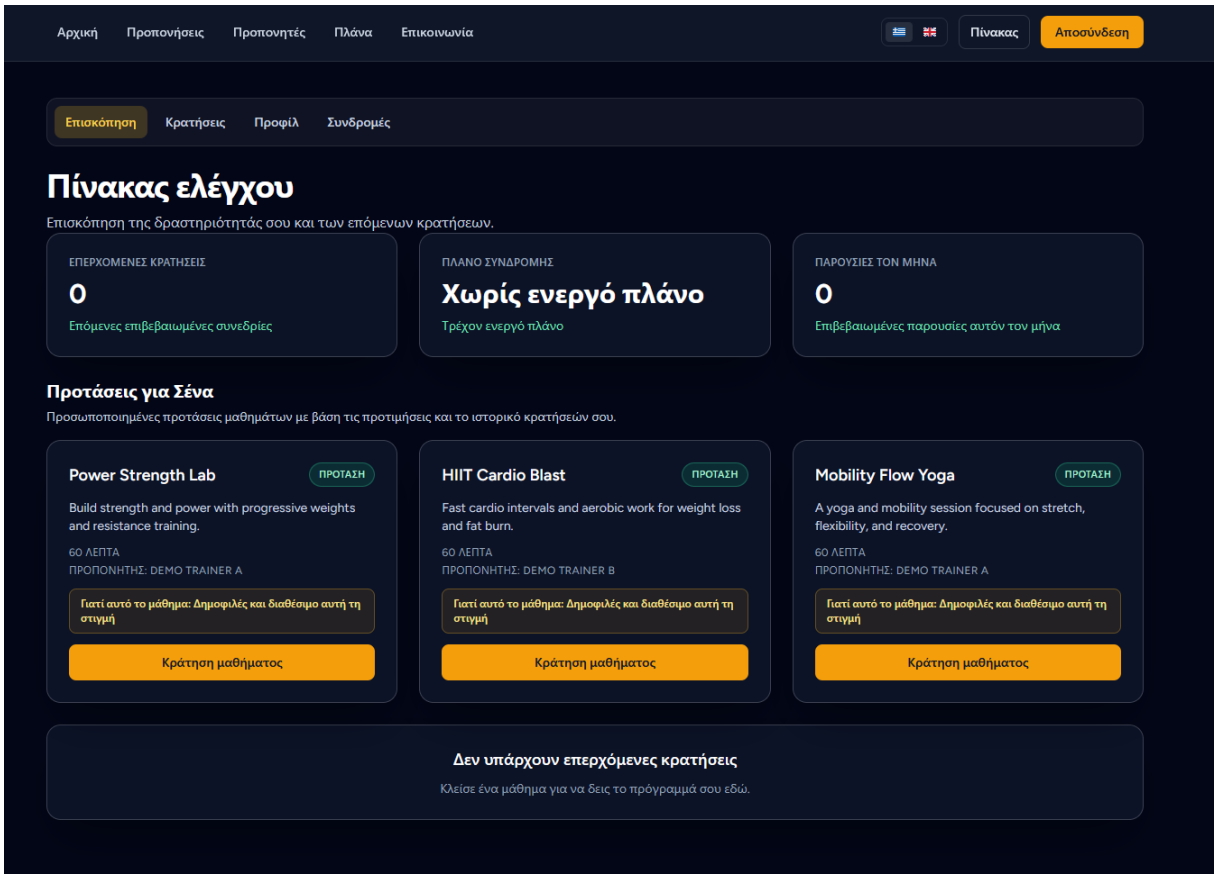


Εικόνα 3.8: Στιγμιότυπο ανάκτησης λογαριασμού χρήστη

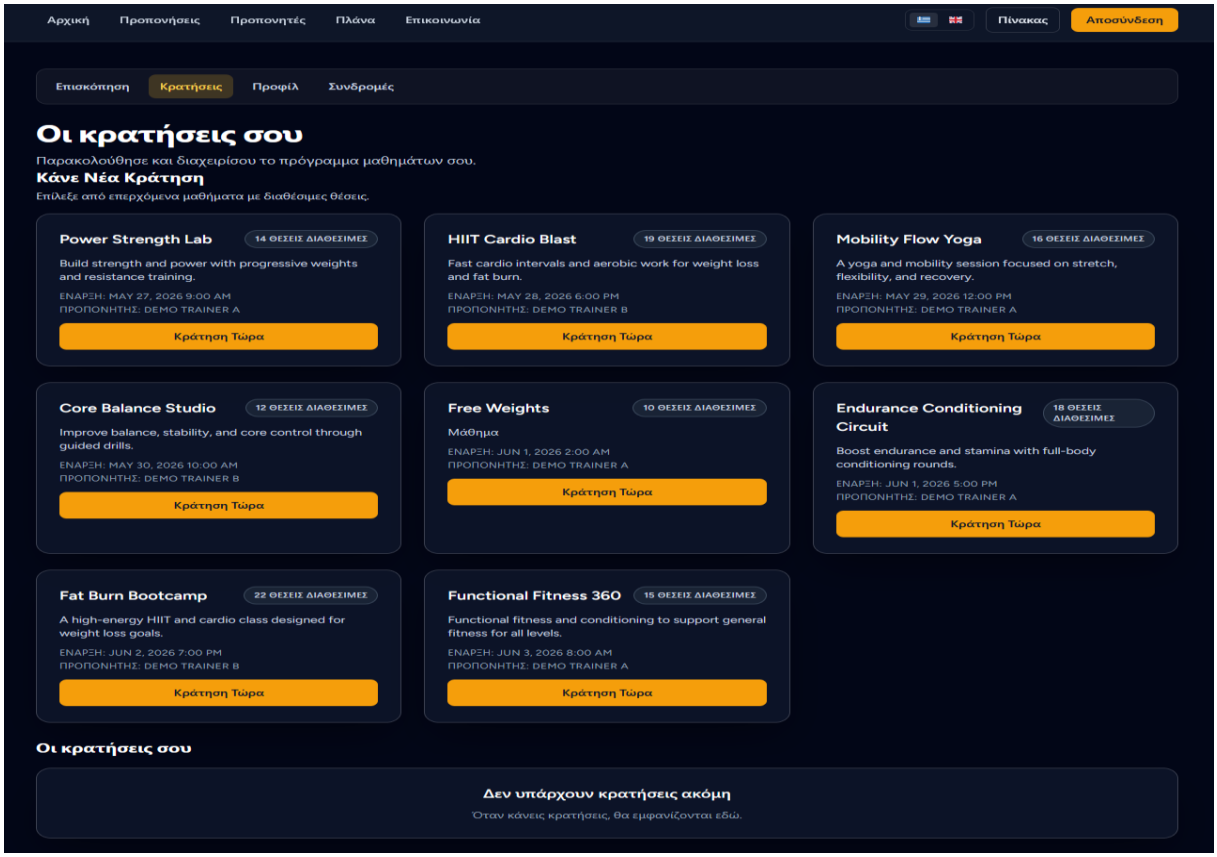
- Σύνδεση: Η οθόνη σύνδεσης ζητά email και κωδικό, με επιλογή “Θυμήσου με” και άμεσο σύνδεσμο για ανάκτηση κωδικού.
- Εγγραφή: Η φόρμα εγγραφής περιλαμβάνει βασικά πεδία (όνομα, email, κωδικό και επιβεβαίωση), ώστε η δημιουργία νέου λογαριασμού να είναι γρήγορη και απλή.
- Ανάκτηση κωδικού: Η οθόνη “Ξεχάσατε τον κωδικό σας;” επιτρέπει την αποστολή email επαναφοράς με ένα μόνο πεδίο, ελαχιστοποιώντας τα βήματα για τον χρήστη.

3.3 Πίνακας ελέγχου χρήστη (dashboard)

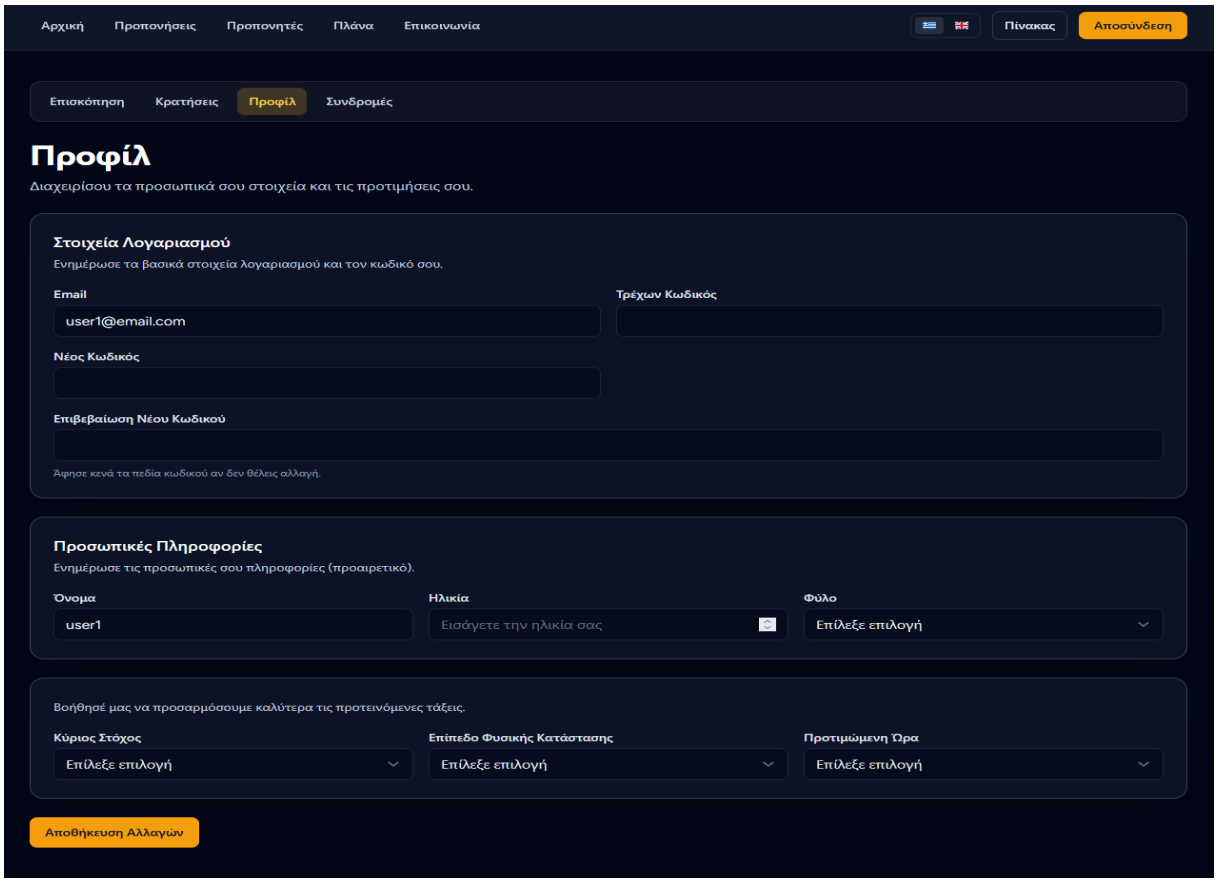
Η υποενότητα 3.3 παρουσιάζει τον πίνακα ελέγχου χρήστη, όπου ο συνδεδεμένος χρήστης διαχειρίζεται το προφίλ, τις κρατήσεις και τη συνδρομή του. Η διεπαφή οργανώνεται σε καρτέλες για άμεση πρόσβαση στις βασικές λειτουργίες και διατηρεί οπτική συνέχεια με το δημόσιο τμήμα του site.



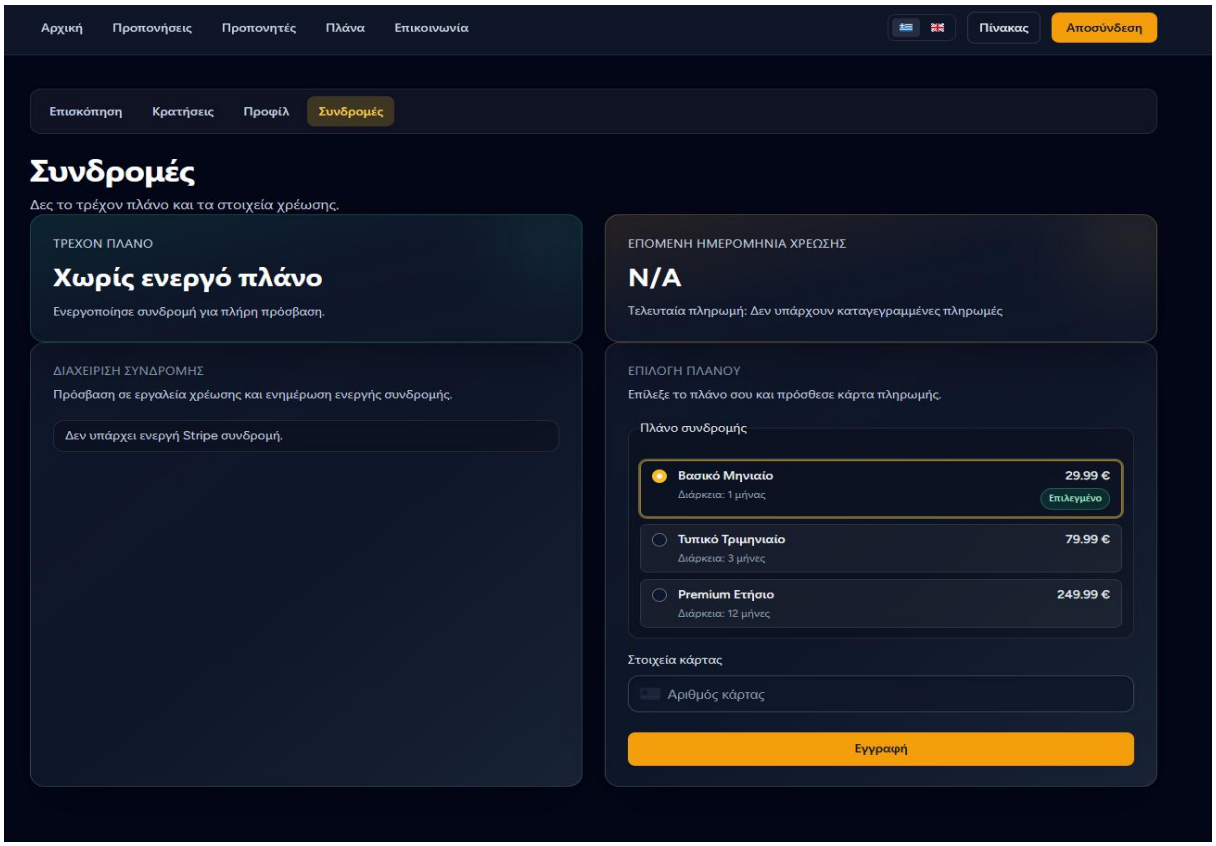
Εικόνα 3.9: Στιγμιότυπο ιστοσελίδας πίνακα ελέγχου επισκόπησης



Εικόνα 3.10: Στιγμιότυπο ιστοσελίδας κρατήσεων χρήστη



Εικόνα 3.11: Στιγμιότυπο σελίδας προφίλ χρήστη

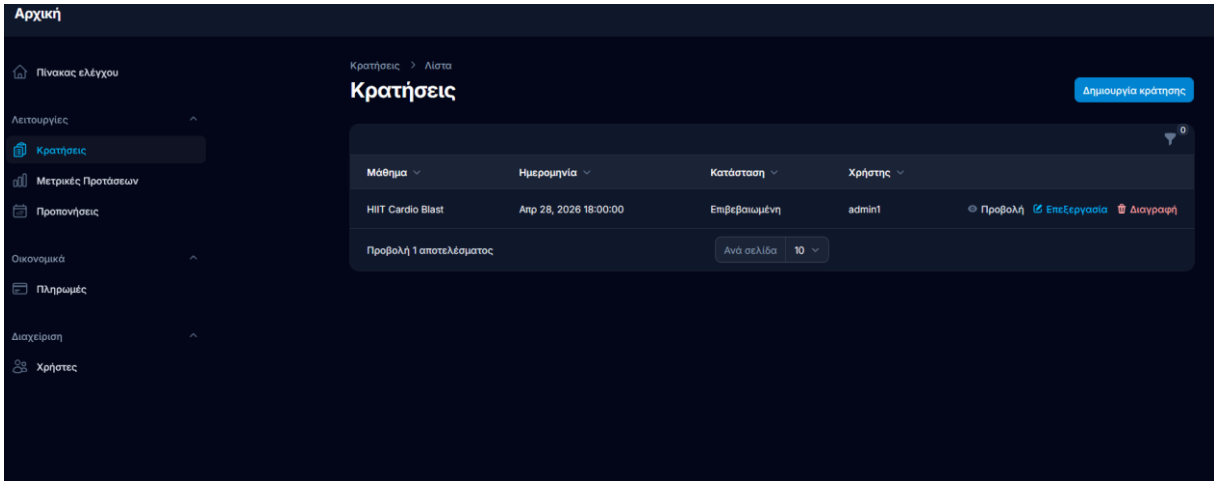


Εικόνα 3.12: Στιγμιότυπο σελίδας αγοράς και προβολής συνδρομής

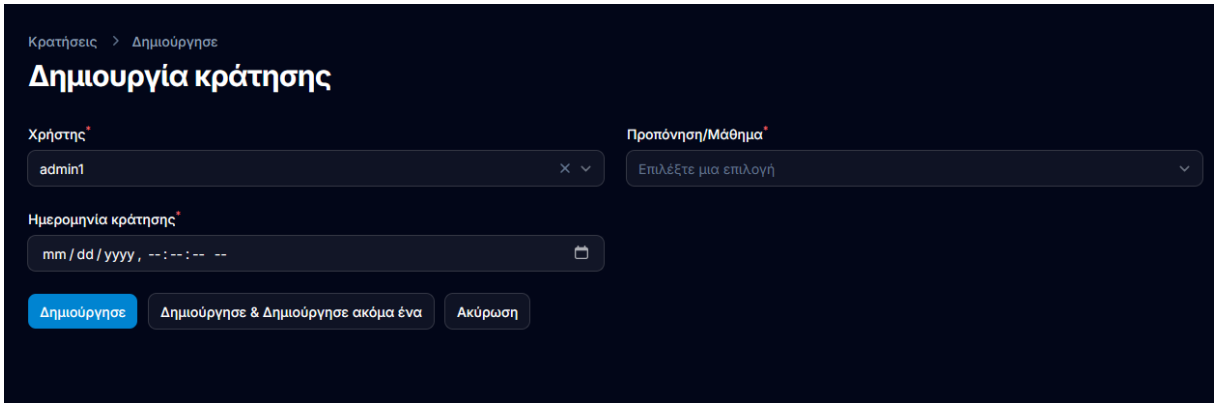
- **Επισκόπηση**: Στην εικόνα 3.9 βλέπουμε συνοπτική εικόνα δραστηριότητας (επερχόμενες κρατήσεις, ενεργό πλάνο, παρουσίες) και ενότητα προτάσεων προπονήσεων/μαθημάτων με δυνατότητα άμεσης κράτησης.
- **Κρατήσεις**: Στην εικόνα 3.10 έχουμε λίστα διαθέσιμων επιλογών με πληροφορίες ώρας, διάρκειας, προπονητή και διαθεσιμότητας, καθώς και περιοχή που εμφανίζει τις ήδη καταχωρισμένες κρατήσεις.
- **Προφίλ**: Στην εικόνα 3.11 παρουσιάζεται επεξεργασία στοιχείων λογαριασμού και βασικών προσωπικών πληροφοριών, μαζί με προτιμήσεις που βοηθούν στην καλύτερη προσωποποίηση.
- **Συνδρομές**: Στην εικόνα 3.12 έχουμε προβολή τρέχοντος πλάνου, στοιχείων χρέωσης και δυνατότητα επιλογής νέου πλάνου/ενεργοποίησης συνδρομής.

3.4 Πίνακας διαχείρισης (Admin panel)

Η ενότητα 3.4 παρουσιάζει τον πίνακα διαχείρισης που χρησιμοποιείται αποκλειστικά από διαχειριστές. Η διεπαφή οργανώνει τις βασικές λειτουργίες σε πίνακες (lists) και φόρμες δημιουργίας/επεξεργασίας, με αναζήτηση, φίλτρα και γρήγορες ενέργειες. Έτσι, η διαχείριση της πλατφόρμας γίνεται από ένα ενιαίο περιβάλλον.

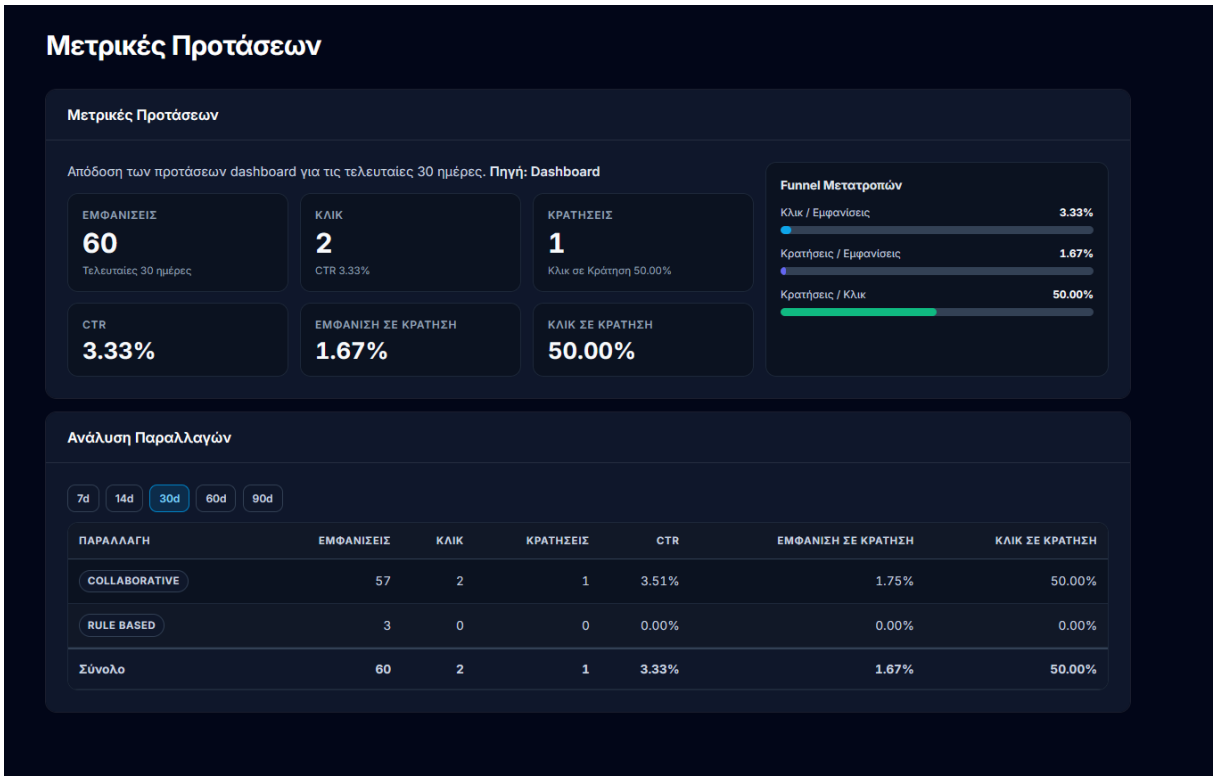


Εικόνα 3.13:Στιγμιότυπο ιστοσελίδας διαχείρισης κρατήσεων



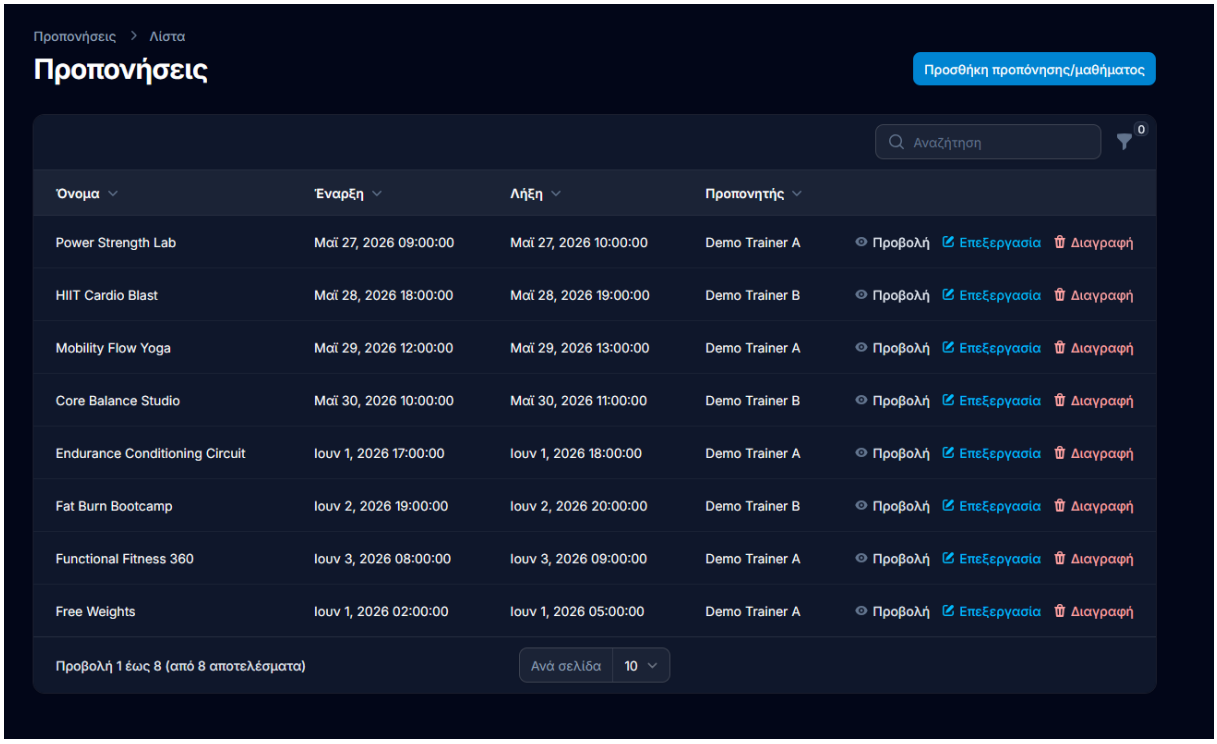
Εικόνα 3.14:Στιγμιότυπα ιστοσελίδας δημιουργίας κράτησης

- **Κρατήσεις (λίστα)**: Επισκόπηση κρατήσεων με μάθημα, ημερομηνία, κατάσταση και χρήστη, για άμεσο έλεγχο της πληρότητας και των επιβεβαιώσεων.
- **Κρατήσεις (δημιουργία)**: Φόρμα εισαγωγής κράτησης που συνδέει χρήστη και μάθημα σε συγκεκριμένη ημερομηνία.

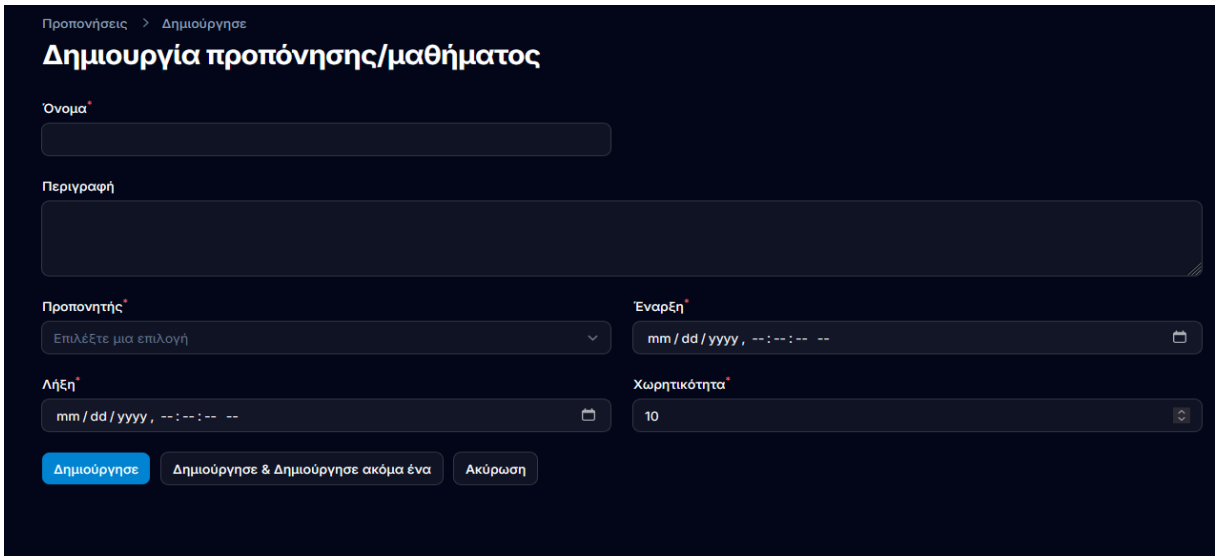


Εικόνα 3.15: Στιγμιότυπο ιστοσελίδας εμφάνισης μετρικών προτάσεων

- **Μετρικές προτάσεων**: Πίνακας στατιστικών που συνοψίζει εμφανίσεις, κλικ, κρατήσεις και ποσοστά μετατροπής, ώστε ο διαχειριστής να αξιολογεί την απόδοση του συστήματος προτάσεων.

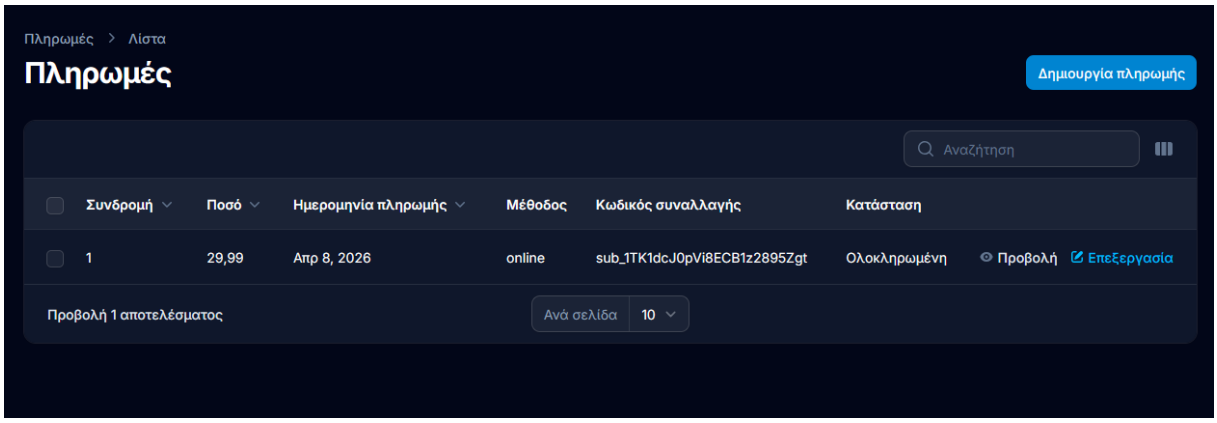


Εικόνα 3.16: Στιγμιότυπο ιστοσελίδας επισκόπησης προπονήσεων/μαθημάτων

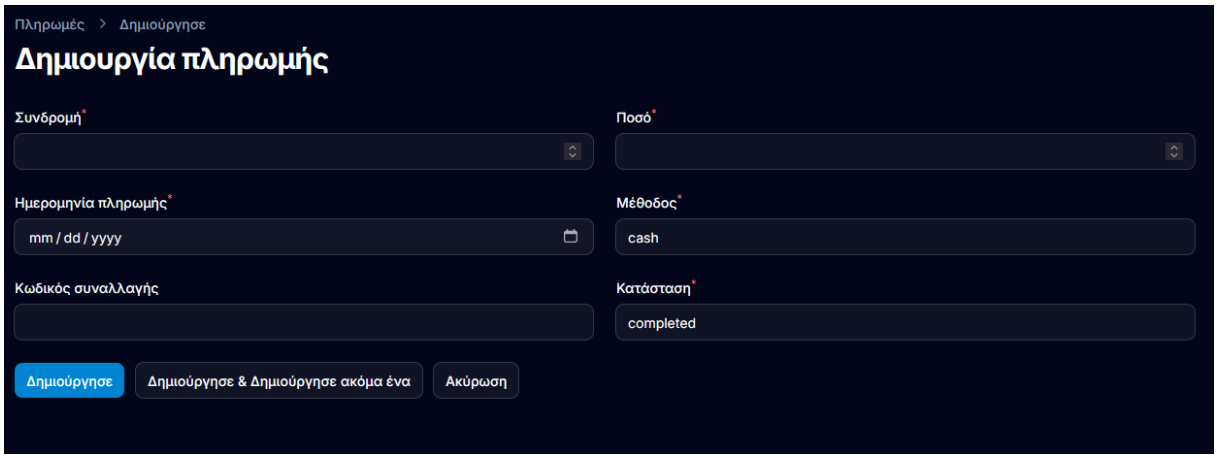


Εικόνα 3.17: Στιγμιότυπο ιστοσελίδας δημιουργίας προπόνησης/μαθήματος

- **Προπονήσεις (λίστα)**: Προβολή όλων των προπονήσεων/μαθημάτων με ώρα έναρξης–λήξης και τον αντίστοιχο προπονητή. Υπάρχουν ενέργειες προβολής, επεξεργασίας και διαγραφής, καθώς και αναζήτηση.
- **Προπονήσεις (δημιουργία)**: Φόρμα εισαγωγής με όνομα, περιγραφή, προπονητή, έναρξη/λήξη και χωρητικότητα. Επιτρέπει γρήγορη δημιουργία νέων προπονήσεων.

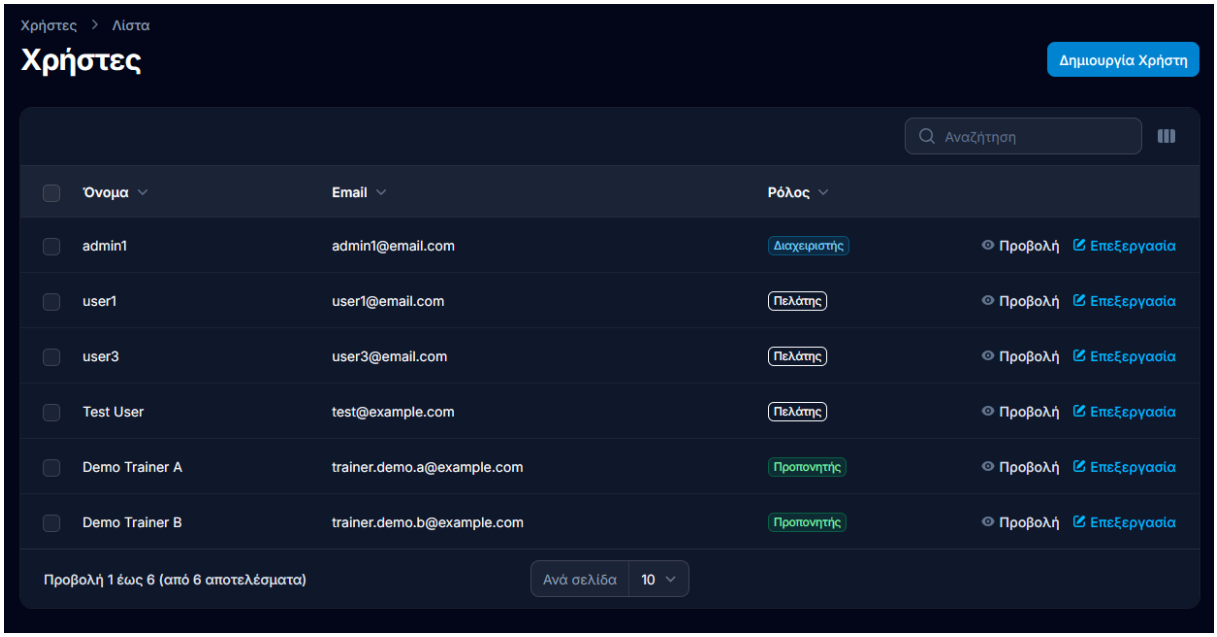


Εικόνα 3.18: Στιγμιότυπο ιστοσελίδας επισκόπησης πληρωμών

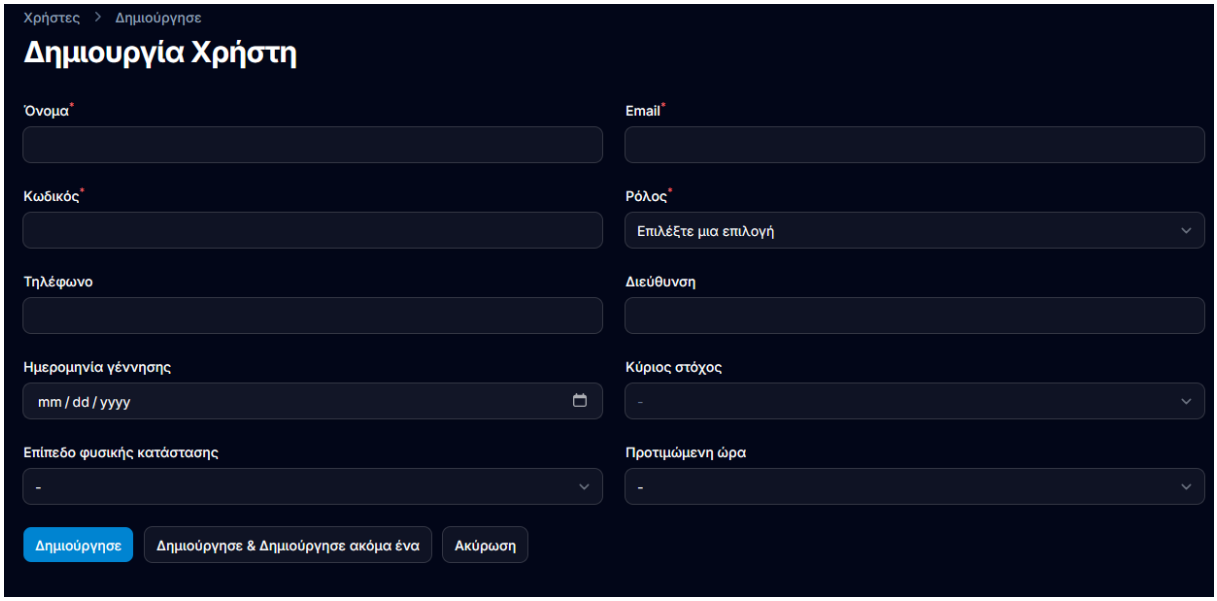


Εικόνα 3.19: Στιγμιότυπο ιστοσελίδας δημιουργίας πληρωμών

- **Πληρωμές (λίστα)**: Συνοπτική εικόνα πληρωμών με ποσό, ημερομηνία, μέθοδο και κατάσταση, ώστε να παρακολουθείται η οικονομική ροή.
- **Πληρωμές (δημιουργία)**: Φόρμα καταχώρησης πληρωμής με βασικά στοιχεία (συνδρομή, ποσό, ημερομηνία, μέθοδος, κατάσταση).



Εικόνα 3.20: Στιγμιότυπο ιστοσελίδας επισκόπησης χρηστών



Εικόνα 3.21: Στιγμιότυπο ιστοσελίδας δημιουργίας χρηστών

- **Χρήστες (λίστα)**: Διαχείριση χρηστών με εμφανή ρόλο (διαχειριστής, πελάτης, προπονητής) και δυνατότητα προβολής/επεξεργασίας.
- **Χρήστες (δημιουργία)**: Φόρμα προσθήκης χρήστη με στοιχεία λογαριασμού και βασικά προφίλ, ώστε να γίνεται γρήγορη καταχώρηση νέων μελών ή προπονητών.

Κεφάλαιο 4ο: Ανάλυση και Αρχιτεκτονική Συστήματος

4.1 Γενική αρχιτεκτονική

Η πλατφόρμα ακολουθεί το πρότυπο MVC του Laravel, με καθαρό διαχωρισμό ανάμεσα σε δρομολόγηση (routes), ελεγκτές (controllers), μοντέλα (models) και προβολές (views). Στο επίπεδο δρομολόγησης γίνεται σαφής διάκριση ανάμεσα στις δημόσιες σελίδες και στο προστατευμένο dashboard, όπου συγκεντρώνονται οι βασικές λειτουργίες του χρήστη (κρατήσεις, προφίλ, συνδρομές). Αυτός ο διαχωρισμός βοηθά τόσο στην ασφάλεια όσο και στην καθαρότητα της ροής.

Στη συνέχεια παρατίθενται επιλεγμένα αποσπάσματα κώδικα, όχι ως πλήρης αναπαραγωγή του έργου, αλλά ως αντιπροσωπευτικά σημεία που δείχνουν πώς υλοποιείται η αρχιτεκτονική στην πράξη. Τα συγκεκριμένα αποσπάσματα επιλέχθηκαν επειδή αποτυπώνουν καθαρά τον διαχωρισμό δημόσιων/προστατευμένων ροών, την οργάνωση του dashboard και τη μεταφορά της σύνθετης λογικής σε services.

4.1.1 Διαχωρισμός δημόσιου site και dashboard(routes)

Η πρώτη υποενότητα δείχνει πώς οργανώνεται η εφαρμογή στο επίπεδο των διαδρομών, ώστε να διαχωρίζεται καθαρά το δημόσιο site από το προστατευμένο dashboard.

```
<?php
use App\Http\Controllers\DashboardPageController;
use App\Http\Controllers\PublicPageController;
use App\Http\Controllers\SubscriptionController;
use App\Http\Middleware\EnsureDashboardAuthenticated;
use Illuminate\Support\Facades\Route;

Route::get('/language/{locale}', function (string $locale) {
    $supported = config('app.supported_locales', ['el', 'en']);
    abort_unless(in_array($locale, $supported, true), 404);
    session(['locale' => $locale]);
    return redirect()->back();
})->name('language.switch');

Route::get('/', [PublicPageController::class, 'home'])->name('home');
Route::get('/classes', [PublicPageController::class, 'classes'])->name('classes.index');
Route::get('/trainers', [PublicPageController::class, 'trainers'])->name('trainers.index');
Route::get('/plans', [PublicPageController::class, 'plans'])->name('plans.index');
Route::get('/contact', [PublicPageController::class, 'contact'])->name('contact.index');
Route::get('/dashboard', [DashboardPageController::class, 'index'])
    ->middleware(EnsureDashboardAuthenticated::class)
    ->name('dashboard');
```

```
Route::prefix('dashboard')
    ->name('dashboard.')
    ->middleware(EnsureDashboardAuthenticated::class)
    ->group(function (): void {
        Route::get('/bookings', [DashboardPageController::class, 'bookings'])->name('bookings');
        Route::post('/bookings', [DashboardPageController::class, 'storeBooking'])->name('bookings.store');
        Route::post('/bookings/{booking}/cancel', [DashboardPageController::class, 'cancelBooking'])-
>name('bookings.cancel');
        Route::get('/profile', [DashboardPageController::class, 'profile'])->name('profile');
        Route::put('/profile', [DashboardPageController::class, 'updateProfile'])->name('profile.update');
        Route::post('/recommendations/click', [DashboardPageController::class, 'trackRecommendationClick'])-
>name('recommendations.click');
        Route::get('/subscriptions', [DashboardPageController::class, 'subscriptions'])->name('subscriptions');
        Route::post('/subscriptions', [SubscriptionController::class, 'store'])->name('subscriptions.store');
        Route::post('/subscriptions/cancel', [SubscriptionController::class, 'destroy'])->name('subscriptions.cancel');
        Route::post('/subscriptions/resume', [SubscriptionController::class, 'resume'])->name('subscriptions.resume');
        Route::post('/subscriptions/portal', [SubscriptionController::class, 'billingPortal'])->name('subscriptions.portal');
    });
require __DIR__.'/auth.php';
```

Ο κώδικας ξεκινάει με τα use statements, τα οποία δηλώνουν ποιοι controllers και middleware χρησιμοποιούνται. Αυτό είναι κρίσιμο για τη δρομολόγηση, γιατί κάθε route πρέπει να «δείχνει» σε συγκεκριμένη μέθοδο controller.

Η διαδρομή Route::get('/language/{locale}', ...) είναι δημόσια και δεν απαιτεί αυθεντικοποίηση. Δέχεται δυναμική παράμετρο locale και ελέγχει αν ανήκει στις υποστηριζόμενες γλώσσες. Αν όχι, επιστρέφει 404. Αν ναι, αποθηκεύει τη γλώσσα στη συνεδρία (session) και κάνει redirect()->back(), ώστε ο χρήστης να παραμείνει στη σελίδα που ήταν.

Στη συνέχεια ορίζονται οι δημόσιες σελίδες: /, /classes, /trainers, /plans, /contact. Κάθε route αντιστοιχίζεται σε συγκεκριμένη μέθοδο του PublicPageController. Αυτό σημαίνει ότι όλες οι δημόσιες λειτουργίες συγκεντρώνονται σε έναν controller και διατηρούνται ανεξάρτητες από το dashboard.

Το /dashboard ορίζεται ξεχωριστά και συνοδεύεται από ->middleware(EnsureDashboardAuthenticated::class). Το middleware λειτουργεί ως φίλτρο πριν εκτελεστεί ο controller: αν ο χρήστης δεν είναι logged in, δεν επιτρέπεται να δει το dashboard. Αυτή είναι η βασική γραμμή άμυνας.

Το μπλοκ Route::prefix('dashboard')->name('dashboard.')->middleware(...) δημιουργεί μια ομαδοποίηση routes. Ο prefix προσθέτει αυτόματα το /dashboard μπροστά σε όλα τα routes της ομάδας, ενώ το name('dashboard.') δημιουργεί ένα κοινό namespace για τα ονόματα routes (π.χ.

dashboard.bookings, dashboard.profile). Αυτό βοηθά στη συντήρηση και μειώνει τα λάθη, γιατί τα routes δεν χρειάζεται να γράφονται ξανά και ξανά με επανάληψη.

Μέσα στην ομάδα, βλέπουμε τις βασικές λειτουργίες χρήστη: κρατήσεις, ακυρώσεις, προφίλ, clicks προτάσεων, συνδρομές και διαχείριση Stripe portal. Όλες αυτές οι λειτουργίες μοιράζονται το ίδιο middleware, άρα προστατεύονται ομοιόμορφα. Η χρήση Route::post για ενέργειες (π.χ. κράτηση, ακύρωση) δείχνει ότι πρόκειται για μεταβολές κατάστασης και όχι απλή προβολή.

Τέλος, το require __DIR__.'/auth.php' φορτώνει τις προεγκατεστημένες routes αυθεντικοποίησης (login, register, reset κ.λπ.). Έτσι το σύστημα authentication είναι οργανωμένο σε ξεχωριστό αρχείο, διατηρώντας καθαρό το βασικό web.php.

4.1.2 Δημόσια αρχική σελίδα και συλλογή βασικών στατιστικών

Η δεύτερη υποενότητα παρουσιάζει τον τρόπο με τον οποίο η δημόσια πλευρά της πλατφόρμας συλλέγει βασικά στατιστικά και τα προβάλλει στο UI με ελαφριά λογική.

```
<?php
    public function home(): View
    {
        $activeMembers = Schema::hasTable('members')
            ? Member::whereNotNull('membership_end_date', 'and')
                ->where('membership_end_date', '>=', now()->toDateString(), 'and')
                ->count('*')
            : (Schema::hasTable('users') && Schema::hasColumn('users', 'role')
                ? User::where('role', '=', 'client', 'and')->count('*')
                : 0);
        $weeklyClasses = Schema::hasTable('classes')
            ? GymClass::where('start_time', '>=', now()->startOfWeek(), 'and')
                ->where('start_time', '<=', now()->endOfWeek(), 'and')
                ->count('*')
            : 0;
        $trainerCount = (Schema::hasTable('users') && Schema::hasColumn('users', 'role'))
            ? User::where('role', '=', 'trainer', 'and')->count('*')
            : 0;
        return view('pages.public.home', [
            'stats' => [
                'activeMembers' => $activeMembers,
                'weeklyClasses' => $weeklyClasses,
                'trainerCount' => $trainerCount,
            ],
        ]);
    }
```

Η μέθοδος home() συγκεντρώνει βασικά στατιστικά για την αρχική σελίδα. Η πρώτη μεταβλητή, $activeMembers, υπολογίζεται μόνο αν υπάρχει ο πίνακας members. Σε αυτή την περίπτωση φιλτράρονται τα μέλη που έχουν membership_end_date και είναι ακόμα ενεργά (η ημερομηνία είναι ≥ σήμερα). Αν ο πίνακας δεν υπάρχει, ο κώδικας πέφτει σε εναλλακτική λύση: μετρά τους χρήστες με ρόλο client, εφόσον υπάρχει ο πίνακας users και η στήλη role. Με αυτό τον τρόπο, η σελίδα δεν «σπάει» σε περιβάλλον όπου δεν έχουν τρέξει όλες οι μεταβάσεις βάσης.

Η μεταβλητή $weeklyClasses μετρά τα μαθήματα/προπονήσεις που βρίσκονται μέσα στο χρονικό παράθυρο της τρέχουσας εβδομάδας. Ο έλεγχος start_time από startOfWeek() μέχρι endOfWeek() εξασφαλίζει ότι το στατιστικό αντιστοιχεί σε πραγματικό εβδομαδιαίο πρόγραμμα. Αν ο πίνακας classes δεν υπάρχει, το αποτέλεσμα γίνεται 0. Αντίστοιχα, το $trainerCount μετρά τους trainers από τον πίνακα users, αλλά μόνο αν η στήλη role είναι διαθέσιμη.

Τέλος, τα τρία στατιστικά περνούν στο view pages.public.home μέσα σε ένα απλό array stats. Αυτό κρατά την παρουσίαση καθαρή: το view έχει έτοιμα τα δεδομένα, ενώ η λογική παραμένει αποκλειστικά στον controller. Η χρήση count() αντί για ανάκτηση εγγραφών μειώνει τη μνήμη και βελτιώνει την απόδοση, καθώς η βάση επιστρέφει μόνο αριθμητικά αποτελέσματα.

4.1.3 Οργήστρωση δεδομένων στο dashboard

Η τρίτη υποενότητα εστιάζει στο dashboard, όπου γίνεται συνδυασμός δεδομένων χρήστη και καταγραφή ενεργειών, ώστε να υποστηρίζεται τόσο η προβολή όσο και η ανάλυση συμπεριφοράς

```
<?php
    $member = $hasMembersTable
        ? Member::query()
            ->with('membershipType:id,name')
            ->where('user_id', $user->getKey())
            ->first()
        : null;
    $recommendations = (Schema::hasTable('classes') && $hasBookingsTable)
        ? $this->recommendationService->getForUser($user, limit: 3, source: 'dashboard')
        : collect();
    if (
        Schema::hasTable('recommendation_impressions')
        && $recommendations->isNotEmpty()
    ) {
        $recommendations = $recommendations
            ->values()
            ->map(function (array $recommendation, int $index) use ($user): array {
                $classId = (int) ($recommendation['gymClass']?->id ?? 0);
                if ($classId <= 0) {
                    $recommendation['impression_id'] = null;
                    return $recommendation;}
```

```
            $impression = RecommendationImpression::query()->create([
                'user_id' => $user->getKey(),
                'class_id' => $classId,
                'variant' => $recommendation['variant'] ?? 'rule_based',
                'source' => $recommendation['source'] ?? 'dashboard',
                'rank' => $index + 1,
                'reason_key' => $recommendation['reason_key'] ?? null,
                'shown_at' => now(),
            ]);

            $recommendation['impression_id'] = $impression->id;

            return $recommendation;
        });}
```

Το απόσπασμα του DashboardPageController.php κάνει τρία βασικά πράγματα: (α) φορτώνει το ενεργό μέλος/πρόγραμμα, (β) ζητά προτάσεις από service, και (γ) καταγράφει "impressions" για τις προτάσεις αυτές.

Αρχικά, η μεταβλητή $member φορτώνει το μέλος που αντιστοιχεί στον χρήστη και "κουβαλά" μαζί μόνο τα απαραίτητα πεδία του membershipType (id, name). Αυτό εξυπηρετεί το dashboard ώστε να εμφανίζει το τρέχον πρόγραμμα χωρίς περιττά δεδομένα. Ο έλεγχος $hasMembersTable προστατεύει από σφάλματα σε περιβάλλον όπου ο πίνακας δεν έχει δημιουργηθεί ακόμη.

Στη συνέχεια, ο controller καλεί το RecommendationService μέσω $this->recommendationService->getForUser(...). Το αποτέλεσμα είναι μια συλλογή (collection) με προτάσεις μαθημάτων. Αν δεν υπάρχουν οι πίνακες classes ή bookings, επιστρέφεται κενή συλλογή, άρα ο controller παραμένει ασφαλής και δεν σπάει.

Το πιο κρίσιμο τμήμα είναι η καταγραφή impressions. Γίνεται μόνο όταν υπάρχουν οι πίνακες recommendation_impressions και όταν οι προτάσεις δεν είναι κενές. Κάθε πρόταση περνά από map():

- Ελέγχεται ότι υπάρχει έγκυρο class_id. Αν όχι, το impression_id μένει null.
- Δημιουργείται νέα εγγραφή RecommendationImpression με πεδία όπως variant, source, rank και reason_key.
- Το rank ορίζεται από το index της πρότασης + 1, άρα κρατάται η σειρά εμφάνισης.
- Το impression_id επιστρέφεται μέσα στο ίδιο αντικείμενο πρότασης, ώστε το frontend να μπορεί να το στείλει αργότερα σε click tracking.

Με αυτή τη λογική, το dashboard δεν είναι απλώς "παρουσίαση" αλλά και σημείο συλλογής δεδομένων αξιολόγησης. Κάθε εμφανιζόμενη πρόταση αποκτά το δικό της αποτύπωμα στη βάση, κάτι που είναι απαραίτητο για να μετρηθεί αργότερα η αποτελεσματικότητα (clicks και bookings ανά πρόταση).

4.1.4 Service layer

```
<?php
    public function getForUser(User $user, int $limit = 3, string $source = 'dashboard'): Collection
    {
        if (! Schema::hasTable('classes') || ! Schema::hasTable('bookings')) {
            return collect();
        }
        $variant = $this->variantForUser($user);
        $recommendations = $variant === 'collaborative'
            ? $this->collaborativeRecommendations($user, $limit)
            : $this->ruleBasedRecommendations($user, $limit);
        if ($recommendations->isEmpty()) {
            $recommendations = $this->fallbackPopularRecommendations($user, $limit);
        }
        return $recommendations
            ->take($limit)
            ->values()
            ->map(function (array $item) use ($variant, $source): array {
                /** @var GymClass $class */
                $class = $item['gym_class'];
                return [
                    'gymClass' => $class,
                    'score' => (float) ($item['score'] ?? 0),
                    'reason_key' => (string) ($item['reason_key'] ?? 'popular'),
                    'variant' => $variant,
                    'source' => $source,
                ];
            });
    }
```

Η τέταρτη υποενότητα δείχνει πώς η σύνθετη λογική προτάσεων απομονώνεται σε service, διατηρώντας τον controller καθαρό και την αρχιτεκτονική πιο επεκτάσιμη.

Η μέθοδος getForUser() αποτελεί το “entry point” για τις προτάσεις. Αρχικά γίνεται έλεγχος ύπαρξης των βασικών πινάκων (classes, bookings). Αν δεν υπάρχουν, επιστρέφεται κενή συλλογή — αυτό αποτρέπει σφάλματα σε περιβάλλον χωρίς migrations.

Έπειτα υπολογίζεται το variant μέσω της variantForUser(), που αποφασίζει αν ο χρήστης θα λάβει rule-based ή collaborative προτάσεις. Αυτό λειτουργεί σαν απλή κατανομή χρηστών σε διαφορετικές στρατηγικές, ώστε να μπορεί να συγκριθεί η απόδοσή τους.

Ανάλογα με το variant, καλείται είτε collaborativeRecommendations() είτε ruleBasedRecommendations(). Αν καμία από αυτές δεν δώσει αποτελέσματα, ενεργοποιείται το fallbackPopularRecommendations() ώστε να μην εμφανιστεί κενό στο UI. Αυτό είναι σημαντικό για την εμπειρία χρήστη: ακόμη και χωρίς επαρκές ιστορικό, υπάρχουν πάντα διαθέσιμες προτάσεις.

Στο τέλος, οι προτάσεις "κανονικοποιούνται" σε ενιαία δομή:

- gymClass περιέχει το αντικείμενο μαθήματος,
- score αποθηκεύει τη βαθμολογία,
- reason_key δείχνει τον βασικό λόγο (π.χ. "popular"),
- variant και source αποθηκεύουν μεταδεδομένα για ανάλυση.

Αυτός ο τελικός χάρτης δεδομένων εξασφαλίζει ότι ο controller δεν χρειάζεται να γνωρίζει ποια μέθοδος χρησιμοποιήθηκε. Απλώς λαμβάνει ομοιόμορφο αποτέλεσμα, το οποίο μπορεί να εμφανιστεί ή να καταγραφεί (impressions/clicks) χωρίς επιπλέον λογική

4.1.5 Επιλογή γλώσσας

Η επιλογή γλώσσας υλοποιείται μέσω της διαδρομής /language/{locale}, η οποία ελέγχει ότι η ζητούμενη γλώσσα είναι στις υποστηριζόμενες και στη συνέχεια την αποθηκεύει στο session, ώστε η επιλογή να "μένει" ενεργή. Μετά τον έλεγχο γίνεται επιστροφή στην προηγούμενη σελίδα, ώστε ο χρήστης να συνεχίζει απρόσκοπτα την πλοήγηση. Στις προβολές, όλα τα κείμενα περνούν από translation keys (π.χ. __()), οπότε το ίδιο view μπορεί να αποδοθεί σε διαφορετικές γλώσσες χωρίς αλλαγή στον κώδικα. Αυτό κρατά τη διεπαφή συνεπή και επιτρέπει εύκολη επέκταση με νέες γλώσσες.

```
<?php
Route::get('/language/{locale}', function (string $locale) {
    $supported = config('app.supported_locales', ['el', 'en']);

    abort_unless(in_array($locale, $supported, true), 404);

    session(['locale' => $locale]);

    return redirect()->back();
})->name('language.switch');
```

Η διαδρομή /language/{locale} λειτουργεί ως κεντρικό σημείο αλλαγής γλώσσας. Ο κώδικας ελέγχει αν το locale ανήκει στις υποστηριζόμενες τιμές (π.χ. el, en). Αν όχι, επιστρέφεται 404. Αν ναι, αποθηκεύεται στο session, ώστε η επιλογή να παραμένει ενεργή σε επόμενες σελίδες, και γίνεται redirect()->back() για να μην διακοπεί η πλοήγηση του χρήστη.

```
@section('title', __('site.nav.contact'))  // 'Η χρήση της συνάρτησης σε block με labels
<label class="mb-1 block text-sm font-semibold text-slate-200">{{ __('site.public.contact.full_name') }}</label>
```

Στα views, όλα τα εμφανιζόμενα κείμενα περνούν από τη συνάρτηση __(), η οποία αναζητά το αντίστοιχο translation key στα αρχεία γλώσσας. Έτσι ο ίδιος κώδικας μπορεί να αποδώσει διαφορετικό

κείμενο ανάλογα με τη γλώσσα που έχει επιλεγεί στο session. Αυτό επιτρέπει κεντρική διαχείριση των μεταφράσεων και εύκολη επέκταση σε νέες γλώσσες.

Οι μεταφράσεις υλοποιούνται μέσω λεξικών (dictionaries) σε PHP arrays, οργανωμένα ανά γλώσσα στους φακέλους el και en. Κάθε κείμενο έχει ένα μοναδικό key, π.χ. site.nav.contact. Στα views χρησιμοποιείται η συνάρτηση __() με αυτό το key, και το Laravel αναζητά την αντίστοιχη τιμή στο λεξικό της τρέχουσας γλώσσας. Αν κάποιο key λείπει, επιστρέφεται το ίδιο key, ώστε να εντοπίζονται εύκολα τα κενά. Για κείμενα που εξαρτώνται από ποσότητες, χρησιμοποιείται το trans_choice(), το οποίο επιλέγει σωστή μορφή (πληθυντικός/ενικός) με βάση τον αριθμό. Έτσι η μετάφραση διατηρείται κεντρικά και η διεπαφή μπορεί να αλλάξει γλώσσα χωρίς αλλαγές στον κώδικα των views.

4.2 Ρόλοι χρηστών και δικαιώματα

Η διαχείριση πρόσβασης βασίζεται σε ρόλους χρηστών (admin, trainer, client) και σε πολιτικές (policies) που καθορίζουν τι επιτρέπεται για κάθε ενέργεια. Με αυτό τον τρόπο αποφεύγεται η διάσπαρτη λογική στους controllers και η ασφάλεια συγκεντρώνεται σε συγκεκριμένα σημεία του κώδικα.

Ακολουθούν στοχευμένα αποσπάσματα κώδικα που τεκμηριώνουν την εφαρμογή των ρόλων και των δικαιωμάτων. Επιλέχθηκαν γιατί δείχνουν καθαρά πώς γίνεται ο έλεγχος πρόσβασης (σε επίπεδο χρήστη) και πώς οι πολιτικές ενοποιούν τη λογική εξουσιοδότησης σε ένα κεντρικό σημείο. Έτσι γίνεται σαφές πώς διασφαλίζεται η προστασία των λειτουργιών και των δεδομένων σε όλο το σύστημα.

4.2.1 Έλεγχος ρόλου admin

Το πρώτο βήμα στον έλεγχο πρόσβασης είναι η αναγνώριση του ρόλου του χρήστη. Στην ενότητα 3.2.1 παρουσιάζεται ο κεντρικός μηχανισμός που αποφασίζει αν ένας χρήστης έχει δικαιώματα διαχειριστή. Αυτή η λογική χρησιμοποιείται ως βάση για τις πολιτικές που ακολουθούν.

```
<?php
    public function isAdmin(): bool
    {
        // Αν υπάρχει το πεδίο 'is_admin', προτίμησέ το· αλλιώς χρησιμοποίησε το enum 'role'.
        if (isset($this->is_admin)) {
            return (bool) $this->is_admin;
        }
        if (isset($this->role)) {
            return $this->role === 'admin';
        }
        return false;
    }
```

Η μέθοδος isAdmin() συγκεντρώνει τον έλεγχο διαχειριστικών δικαιωμάτων σε ένα σημείο. Πρώτα ελέγχει αν υπάρχει ξεχωριστό πεδίο is_admin (πιο ρητή επιλογή), και αν δεν υπάρχει, χρησιμοποιεί το πεδίο role ως εναλλακτική. Έτσι ο υπόλοιπος κώδικας μπορεί να καλεί απλά isAdmin() χωρίς να γνωρίζει τις λεπτομέρειες αποθήκευσης του ρόλου.

4.2.2 Πολιτική κρατήσεων (ενημέρωση/διαγραφή)

Σε αυτήν την ενότητα παρουσιάζεται η πολιτική κρατήσεων, η οποία ρυθμίζει ποιος μπορεί να ενημερώνει ή να διαγράφει μια κράτηση. Η λογική αυτή είναι κρίσιμη γιατί συνδυάζει ρόλους (admin, trainer) με ιδιοκτησία δεδομένων, εξασφαλίζοντας ότι κάθε ενέργεια εκτελείται από τον σωστό χρήστη.

```
<?php
   public function update(User $user, Booking $booking): bool
   {
      // Οι διαχειριστές μπορούν να ενημερώνουν οποιαδήποτε κράτηση.
      if ($user->isAdmin()) {
         return true;   }
      // Οι προπονητές μπορούν να ενημερώνουν κρατήσεις για μαθήματα που τους ανήκουν.
      if (isset($user->role) && $user->role === 'trainer') {
         // Αν η σχέση δεν έχει φορτωθεί, κάνε ασφαλή σύγκριση μέσω αναζήτησης στο gym_class.
         if ($booking->relationLoaded('gymClass')) {
            return $booking->gymClass->trainer_id === $user->getKey();  }
         // Σε διαφορετική περίπτωση: φόρτωσε το gymClass και σύγκρινε το trainer_id
         return $booking->gymClass()->where('id', $booking->class_id)->where('trainer_id', $user->getKey())->exists();   }
      return false;}
   public function delete(User $user, Booking $booking): bool
   {
      //Οι διαχειριστές ή ο χρήστης μπορούν να διαγράψουν την κράτηση που τους ανήκει.
      if ($user->isAdmin()) {
         return true;
      }
      return $booking->user_id === $user->getKey();
   }
```

Η ενημέρωση κρατήσεων επιτρέπεται είτε σε admin είτε σε trainer που “κατέχει” το μάθημα της κράτησης. Ο έλεγχος γίνεται σε δύο στάδια: αν έχει ήδη φορτωθεί η σχέση gymClass χρησιμοποιείται απευθείας, αλλιώς γίνεται ασφαλές query στη βάση. Αυτό αποφεύγει λάθη όταν οι σχέσεις δεν έχουν γίνει eager-load. Για τη διαγραφή, επιτρέπεται είτε σε admin είτε στον ίδιο τον χρήστη που έκανε την κράτηση, ώστε να προστατεύονται τα δεδομένα άλλων χρηστών.

4.2.3 Πολιτική προπονήσεων/μαθημάτων (προβολή/ενημέρωση)

Η πολιτική κρατήσεων είναι χαρακτηριστικό παράδειγμα συνδυασμού ρόλων και ιδιοκτησίας δεδομένων. Στον κώδικα παρακάτω παρουσιάζεται πώς ορίζονται οι κανόνες για ενημέρωση και διαγραφή κρατήσεων, ώστε να επιτρέπονται μόνο σε χρήστες που έχουν πραγματική σχέση με το μάθημα ή την κράτηση.

```
<?php
   public function view(User $user, GymClass $gymClass): bool
   {
      if ($user->isAdmin()) {
         return true;
      }
      // Να επιτρέπεται στους χρήστες να βλέπουν ένα μάθημα αν έχουν κράτηση για αυτό.
      return $gymClass->bookings()->where('user_id', $user->getKey())->exists();
   }
   public function update(User $user, GymClass $gymClass): bool
   {
      // Οι διαχειριστές μπορούν να ενημερώνουν οποιοδήποτε μάθημα. Οι προπονητές μπορούν να ενημερώνουν μαθήματα
στα οποία είναι ανατεθειμένοι.
      if ($user->isAdmin()) {
         return true;
      }
      // Να επιτρέπεται στους χρήστες-προπονητές να ενημερώνουν τα μαθήματα που τους ανήκουν.
      if (isset($user->role) && $user->role === 'trainer') {
         return $gymClass->trainer_id === $user->getKey();
      }
      return false;
   }
```

Η μέθοδος update() επιτρέπει χωρίς περιορισμούς την ενημέρωση σε admin, γιατί ο ρόλος αυτός έχει πλήρη πρόσβαση. Για τους trainers, ο έλεγχος δεν βασίζεται μόνο στον ρόλο αλλά και στην ιδιοκτησία του μαθήματος: ο trainer μπορεί να ενημερώσει κράτηση μόνο αν το μάθημα ανήκει σε αυτόν. Αυτό επιτυγχάνεται με δύο τρόπους:

- Αν η σχέση gymClass έχει ήδη φορτωθεί, γίνεται άμεση σύγκριση του trainer_id.
- Αν όχι, γίνεται ασφαλής αναζήτηση στη βάση, ώστε να μην υπάρξει λάθος λόγω απουσίας σχέσης.

Η μέθοδος delete() ακολουθεί πιο «ιδιοκτησιακό» μοντέλο: επιτρέπεται είτε στον admin είτε στον ίδιο τον χρήστη που έκανε την κράτηση. Έτσι αποτρέπεται ένας χρήστης να διαγράψει κράτηση άλλου, ενώ παραμένει δυνατή η διαχειριστική παρέμβαση όταν χρειάζεται.

Συνολικά, η πολιτική κρατήσεων δείχνει πώς οι ρόλοι (admin/trainer) συνδυάζονται με έλεγχο ιδιοκτησίας (user_id, trainer_id) για να δημιουργηθεί σαφές και ασφαλές πλαίσιο πρόσβασης.

4.2.4 Σύνδεση μοντέλων με πολιτικές

Στην ενότητα 4.2.4 παρουσιάζεται η χαρτογράφηση των μοντέλων στις αντίστοιχες πολιτικές τους. Αυτό το βήμα είναι κρίσιμο, γιατί ενεργοποιεί επίσημα τους κανόνες πρόσβασης σε όλο το σύστημα και εξασφαλίζει ότι κάθε μοντέλο προστατεύεται από τη σωστή πολιτική.

```
<?php
   protected $policies = [
      Payment::class => PaymentPolicy::class,
      User::class => UserPolicy::class,
      GymClass::class => GymClassPolicy::class,
      Booking::class => BookingPolicy::class,
   ];

   public function boot(): void
   {
      $this->registerPolicies();
   }
```

Ο AuthServiceProvider χαρτογραφεί κάθε μοντέλο στην αντίστοιχη πολιτική του. Αυτό σημαίνει ότι όταν το σύστημα κάνει authorize ή can, γνωρίζει αυτόματα ποια policy να χρησιμοποιήσει. Το registerPolicies() ενεργοποιεί αυτή τη χαρτογράφηση στο Laravel, διασφαλίζοντας ότι η πολιτική εφαρμόζεται ομοιόμορφα σε όλη την εφαρμογή.

4.3 Βάση δεδομένων και βασικές οντότητες

Πριν παρουσιαστούν οι βασικοί πίνακες, χρειάζεται μια σύντομη αναφορά στον ρόλο των migrations στο Laravel. Οι migrations είναι αρχεία κώδικα που περιγράφουν βήμα-βήμα το σχήμα της βάσης (πίνακες, πεδία, σχέσεις και περιορισμούς). Λειτουργούν σαν "έκδοση" της βάσης δεδομένων: κάθε αλλαγή καταγράφεται σε ξεχωριστό migration, ώστε να μπορεί να εφαρμοστεί αυτόματα σε οποιοδήποτε περιβάλλον (τοπικό, staging, παραγωγή) χωρίς χειροκίνητες αλλαγές. Με αυτό τον τρόπο εξασφαλίζεται ότι όλοι οι developers και όλοι οι servers έχουν το ίδιο schema, ενώ υπάρχει δυνατότητα rollback αν κάτι πάει λάθος. Επομένως οι migrations δεν είναι απλώς τεχνικό εργαλείο, αλλά βασικό στοιχείο για σταθερότητα και συνέπεια στην ανάπτυξη.

Η βάση δεδομένων οργανώνεται γύρω από τον κύκλο ζωής του μέλους και τις λειτουργίες του γυμναστηρίου (προπονήσεις, κρατήσεις, συνδρομές, πληρωμές). Στα παρακάτω αποσπάσματα φαίνονται οι βασικοί πίνακες και οι σχέσεις τους, ώστε να γίνει σαφές πώς το σχήμα υποστηρίζει τις κύριες ροές.

Ακολουθούν στοχευμένα αποσπάσματα από migrations και μοντέλα. Επιλέχθηκαν γιατί αποτυπώνουν τις κρίσιμες οντότητες (χρήστες, προπόνησεις, κρατήσεις, συνδρομές, πληρωμές) και τον τρόπο που δένουν μεταξύ τους τόσο σε επίπεδο βάσης όσο και σε επίπεδο ORM.

4.3.1 Πίνακας χρηστών και ρόλοι

Ξεκινάμε από τον πίνακα χρηστών, επειδή αποτελεί τη βάση για όλες τις υπόλοιπες σχέσεις και τα δικαιώματα πρόσβασης.

```
<?php
      Schema::create('users', function (Blueprint $table) {
         $table->id();
         $table->string('name');
```

```
        $table->string('email')->unique();
        $table->timestamp('email_verified_at')->nullable();
        $table->string('password');
        $table->enum('role', ['admin', 'trainer', 'client'])->default('client');
        $table->string('phone')->nullable();
        $table->string('address')->nullable();
        $table->date('birth_date')->nullable();
        $table->rememberToken();
        $table->timestamps();
    });
```

Ο πίνακας users περιλαμβάνει τα βασικά στοιχεία ταυτοποίησης και τον ρόλο (role). Αυτό επιτρέπει το σύστημα να διαφοροποιεί δικαιώματα (admin, trainer, client) από το επίπεδο της βάσης. Τα πεδία phone, address, birth_date επιτρέπουν εμπλουτισμένο προφίλ χρήστη, ενώ το rememberToken υποστηρίζει παραμονή σύνδεσης.

4.3.2 Προπονήσεις/Μαθήματα

Στη συνέχεια παρουσιάζονται οι προπονήσεις/μαθήματα, ως η βασική οντότητα πάνω στην οποία στηρίζονται οι κρατήσεις και το πρόγραμμα του γυμναστηρίου.

```
<?php
    Schema::create('classes', function (Blueprint $table) {
        $table->id();
        $table->string('name'); // όνομα μαθήματος
        $table->text('description')->nullable();
        $table->foreignId('trainer_id')->constrained('users')->onDelete('cascade');
        $table->dateTime('start_time');
        $table->dateTime('end_time');
        $table->integer('capacity')->default(10); // μέγιστος αριθμός συμμετεχόντων
        $table->timestamps();    });
```

Ο πίνακας classes περιγράφει κάθε προπόνηση/μάθημα με χρονικά όρια και χωρητικότητα. Το trainer_id συνδέεται απευθείας με τον πίνακα users, άρα κάθε προπόνηση/μάθημα έχει “ιδιοκτήτη” trainer. Η χωρητικότητα (capacity) χρησιμοποιείται αργότερα για να φιλτράρονται οι διαθέσιμες κρατήσεις.

4.3.3 Κρατήσεις

Εδώ αποτυπώνεται ο μηχανισμός που συνδέει χρήστες και προπονήσεις/μαθήματα, δηλαδή το σημείο όπου η λειτουργικότητα γίνεται πραγματική ενέργεια

```
<?php
    Schema::create('bookings', function (Blueprint $table) {
        $table->id();
```

```
        $table->foreignId('user_id')->constrained()->onDelete('cascade');
        $table->foreignId('class_id')->constrained()->onDelete('cascade');
        $table->dateTime('booking_date'); // πότε έκανε κράτηση
        $table->enum('status', ['confirmed', 'cancelled'])->default('confirmed');
        $table->timestamps();
    });
```

Κάθε κράτηση συνδέει έναν χρήστη με ένα μάθημα μέσω user_id και class_id. Το status επιτρέπει να διακρίνονται ενεργές από ακυρωμένες κρατήσεις, χωρίς να χάνεται το ιστορικό. Το booking_date αποθηκεύει το χρονικό σημείο της κράτησης, ώστε να μπορούν να γίνουν έλεγχοι "παρελθόντος/μέλλοντος".

4.3.4 Μέλη και τύποι συνδρομών

Η ενότητα αυτή καλύπτει το προφίλ μέλους και τα διαθέσιμα πακέτα, που μαζί περιγράφουν τον κύκλο ζωής μιας συνδρομής. Σε αυτό το σημείο γίνεται και η διάκριση ανάμεσα στον πίνακα users (λογαριασμοί πρόσβασης, ρόλοι, στοιχεία σύνδεσης) και στον πίνακα members (στοιχεία μέλους, ημερομηνίες συνδρομής, στόχοι και προτιμήσεις). Ο διαχωρισμός αυτός επιτρέπει να υπάρχουν χρήστες που δεν είναι ενεργά μέλη (π.χ. admin, trainer), ενώ ταυτόχρονα διατηρείται ένα πιο "πλούσιο" προφίλ για τους πελάτες του γυμναστηρίου.

```
<?php
    Schema::create('members', function (Blueprint $table): void {
        $table->id();
        $table->foreignId('user_id')->nullable()->constrained()->nullOnDelete();
        $table->string('name');
        $table->string('email')->nullable();
        $table->string('phone')->nullable();
        $table->string('address')->nullable();
        $table->enum('sex', ['male', 'female', 'other'])->nullable();
        $table->date('membership_start_date')->nullable();
        $table->date('membership_end_date')->nullable();
        $table->unsignedBigInteger('membership_type_id')->nullable();
        $table->timestamps();
    });
```

Ο πίνακας members λειτουργεί ως επαγγελματικό "προφίλ μέλους" και συνδέεται (προαιρετικά) με τον user. Περιλαμβάνει στοιχεία συνδρομής (ημερομηνίες, τύπος συνδρομής), τα οποία είναι απαραίτητα για τον κύκλο ζωής της συμμετοχής.

```
<?php
    Schema::create('membership_types', function (Blueprint $table) {
        $table->id();
        $table->foreignId('user_id')->nullable()->constrained()->nullOnDelete();
```

```
        $table->string('name');
        $table->text('description')->nullable();
        $table->decimal('price', 8, 2)->default(0);
        $table->integer('duration_months')->default(1);
        $table->string('stripe_price_id')->nullable();
        $table->timestamps();
    });
```

Ο πίνακας membership_types αποθηκεύει τα διαθέσιμα πακέτα, με διάρκεια και τιμή. Επιπλέον, στο membership_types προστέθηκε το πεδίο stripe_price_id μέσω migration στο πλαίσιο της ενσωμάτωσης του Cashier, ώστε κάθε πακέτο να αντιστοιχεί σε συγκεκριμένο Price του Stripe.

4.3.5 Συνδρομές και πληρωμές

Σε αυτή την υποενότητα παρουσιάζεται ο τρόπος με τον οποίο το σύστημα αποθηκεύει τις συνδρομές και τις πληρωμές. Η υλοποίηση βασίζεται στο Laravel Cashier, το οποίο εισάγει/χρησιμοποιεί πεδία που αντιστοιχούν σε αντικείμενα του Stripe (π.χ. stripe_id, stripe_status, stripe_price), ώστε η κατάσταση της συνδρομής να συγχρονίζεται με το εξωτερικό σύστημα πληρωμών.

4.3.5.1 Σημείωση για cashier

Στο πλαίσιο της ενσωμάτωσης, το Cashier εισάγει/χρησιμοποιεί τα παραπάνω πεδία στους πίνακες subscriptions και subscription_items, ώστε η εφαρμογή να έχει πλήρη εικόνα της κατάστασης συνδρομών όπως τις διαχειρίζεται το Stripe. Συνδυαστικά με το stripe_price_id του membership_types (βλ. 3.3.4), η εφαρμογή μπορεί να δημιουργεί και να ενημερώνει συνδρομές με συνέπεια.

```
<?php
    Schema::create('subscriptions', function (Blueprint $table) {
        $table->id();
        $table->foreignId('user_id')->constrained()->onDelete('cascade');
        $table->string('type');
        $table->string('stripe_id')->unique();
        $table->string('stripe_status');
        $table->string('stripe_price')->nullable();
        $table->integer('quantity')->nullable();
        $table->timestamp('trial_ends_at')->nullable();
        $table->timestamp('ends_at')->nullable();
        $table->date('start_date')->nullable();
        $table->date('end_date')->nullable();
        $table->decimal('price', 8, 2)->default(0);
        $table->enum('status', ['active', 'expired'])->default('active');
        $table->timestamps();
        $table->index(['user_id', 'stripe_status']);  });
```

Ο πίνακας subscriptions κρατά τα στοιχεία της συνδρομής όπως επιστρέφονται από το Stripe μέσω Cashier. Το stripe_id αντιστοιχεί στη συνδρομή στο Stripe, ενώ το stripe_status και το stripe_price αποθηκεύουν την κατάστασή της και το Price που χρεώνεται. Τα πεδία trial_ends_at, ends_at, start_date, end_date επιτρέπουν στην εφαρμογή να γνωρίζει το χρονικό πλαίσιο της συνδρομής χωρίς να χρειάζεται συνεχές API call στο Stripe. Το status (active/expired) κρατά ένα επιπλέον "εσωτερικό" state για business χρήση.

```
<?php
    Schema::create('subscription_items', function (Blueprint $table) {
        $table->id();
        $table->foreignId('subscription_id');
        $table->string('stripe_id')->unique();
        $table->string('stripe_product');
        $table->string('stripe_price');
        $table->integer('quantity')->nullable();
        $table->timestamps();

        $table->index(['subscription_id', 'stripe_price']);
    });
```

Ο πίνακας subscription_items αποτυπώνει τα επιμέρους items μιας συνδρομής. Το Cashier χρησιμοποιεί αυτά τα δεδομένα για να συσχετίζει προϊόντα/τιμές του Stripe με την εσωτερική συνδρομή. Έτσι υποστηρίζονται πιο σύνθετα σενάρια (π.χ. αλλαγή price, πολλαπλά items) χωρίς να χαθεί η συνέπεια.

```
<?php
    Schema::create('payments', function (Blueprint $table) {
        $table->id();
        $table->foreignId('subscription_id')->constrained()->onDelete('cascade');
        $table->foreignId('user_id')->nullable()->constrained()->nullOnDelete();
        $table->unsignedBigInteger('member_id')->nullable();
        $table->decimal('amount', 8, 2);
        $table->date('payment_date');
        $table->enum('method', ['cash', 'card', 'online'])->default('cash');
        $table->string('payment_method')->nullable();
        $table->string('transaction_id')->nullable();
        $table->enum('status', ['completed', 'pending', 'failed'])->default('completed');
        $table->timestamps();
    });
```

Ο πίνακας payments καταγράφει το οικονομικό ιστορικό: ποσό, ημερομηνία, μέθοδο πληρωμής και κατάσταση. Η σύνδεση με subscription_id επιτρέπει να γνωρίζουμε για ποια συνδρομή έγινε κάθε πληρωμή, ενώ τα πεδία payment_method και transaction_id εξυπηρετούν online πληρωμές (Stripe).

Έτσι, η εφαρμογή διατηρεί δικό της ιστορικό, ανεξάρτητα από το Stripe, χρήσιμο για αναφορές και audit.

4.3.6 Tracking προτάσεων

Τέλος παρουσιάζεται το σχήμα παρακολούθησης προτάσεων, ώστε να είναι μετρήσιμη η αποτελεσματικότητα του recommendation module.

```
<?php
        Schema::create('recommendation_impressions', function (Blueprint $table): void {
            $table->id();
            $table->foreignId('user_id')->constrained()->cascadeOnDelete();
            $table->foreignId('class_id')->constrained('classes')->cascadeOnDelete();
            $table->string('variant', 50);
            $table->string('source', 50)->nullable();
            $table->unsignedSmallInteger('rank')->default(1);
            $table->string('reason_key', 50)->nullable();
            $table->timestamp('shown_at')->useCurrent();
            $table->timestamps();

            $table->index(['user_id', 'shown_at']);
            $table->index(['variant', 'source']);});
```

```
<?php
        Schema::create('recommendation_clicks', function (Blueprint $table): void {
            $table->id();
            $table->foreignId('impression_id')->constrained('recommendation_impressions')->cascadeOnDelete();
            $table->foreignId('user_id')->constrained()->cascadeOnDelete();
            $table->timestamp('clicked_at')->useCurrent();
            $table->timestamps();
            $table->index(['user_id', 'clicked_at']);  });
```

```
<?php
        Schema::create('booking_attributions', function (Blueprint $table): void {
            $table->id();
            $table->foreignId('booking_id')->constrained()->cascadeOnDelete();
            $table->foreignId('impression_id')->nullable()->constrained('recommendation_impressions')-> nullOnDelete();
            $table->foreignId('click_id')->nullable()->constrained('recommendation_clicks')->nullOnDelete();
            $table->foreignId('user_id')->constrained()->cascadeOnDelete();
```

```
            $table->foreignId('class_id')->constrained('classes')->cascadeOnDelete();
            $table->string('variant', 50)->nullable();
            $table->string('source', 50)->nullable();
            $table->timestamp('booked_at')->useCurrent();
            $table->timestamps();
            $table->unique('booking_id');
            $table->index(['variant', 'source']);
        });
```

Οι πίνακες παρακολούθησης (impressions, clicks, booking_attributions) δημιουργούν μια πλήρη αλυσίδα αξιολόγησης των προτάσεων. Το recommendation_impressions καταγράφει κάθε εμφάνιση πρότασης και κρατά το πλαίσιο της: variant (ποια στρατηγική χρησιμοποιήθηκε), source (από ποια σελίδα), rank (σειρά εμφάνισης) και reason_key (λόγος πρότασης). Έτσι γνωρίζουμε όχι μόνο τι εμφανίστηκε, αλλά και πώς/γιατί εμφανίστηκε.

Το recommendation_clicks καταγράφει κάθε ενεργή αλληλεπίδραση του χρήστη με μια πρόταση και συνδέεται απευθείας με το impression. Αυτό επιτρέπει την εξαγωγή δεικτών όπως CTR (click-through rate) ανά στρατηγική, ανά πηγή, ή ανά θέση στη λίστα.

Το booking_attributions συνδέει την τελική κράτηση με το αντίστοιχο impression και click, ώστε να μπορούμε να μετρήσουμε πραγματικές μετατροπές (conversions). Η ύπαρξη unique(booking_id) αποτρέπει διπλομετρήσεις και κρατά καθαρή την αναλυτική εικόνα.

Συνολικά, το σχήμα αυτό δεν είναι απλώς “logging”, αλλά υποστηρίζει ουσιαστική αξιολόγηση του recommendation module (CTR, conversion rate, απόδοση ανά variant), κάτι που είναι κρίσιμο για το Κεφάλαιο 5.

4.4 Ροές: κρατήσεις, συνδρομές, πληρωμές

Σε αυτό το υποκεφάλαιο παρουσιάζονται οι βασικές ροές του συστήματος όπως υλοποιούνται στο backend: δημιουργία/ακύρωση κρατήσεων, δημιουργία συνδρομών μέσω Stripe, και καταγραφή πληρωμών. Τα αποσπάσματα που ακολουθούν δείχνουν τα “κρίσιμα σημεία” όπου εφαρμόζονται έλεγχοι, ενημερώνεται η βάση και εξασφαλίζεται συνέπεια δεδομένων.

4.4.1 Ροή κράτησης – Έλεγχοι και δημιουργία

Ξεκινάμε με τη βασική ροή κράτησης, γιατί αποτελεί την κύρια ενέργεια του χρήστη μέσα στην πλατφόρμα.

4.4.1.1 Έλεγχοι διαθεσιμότητας/διπλής κράτησης

```
<?php
    public function storeBooking(Request $request): RedirectResponse
    {
        if (! Schema::hasTable('bookings') || ! Schema::hasTable('classes')) {
            return redirect()
                ->route('dashboard.bookings')
                ->with('status', __('site.dashboard.bookings.unavailable'));
        }
```

```
$validated = $request->validate([
    'class_id' => ['required', 'integer', 'exists:classes,id'],
    'recommendation_impression_id' => ['nullable', 'integer'],
]);
$user = $request->user();
$class = GymClass::query()
    ->withCount([
        'bookings as confirmed_bookings_count' => fn ($query) => $query->where('status', 'confirmed'),
    ])
    ->whereKey($validated['class_id'])
    ->first();
if (! $class || ! $class->start_time || $class->start_time->isPast()) {
    return redirect()
        ->route('dashboard.bookings')
        ->withErrors(['class_id' => __('site.dashboard.bookings.class_unavailable')]);
}
if ((int) $class->confirmed_bookings_count >= (int) $class->capacity) {
    return redirect()
        ->route('dashboard.bookings')
        ->withErrors(['class_id' => __('site.dashboard.bookings.class_full')]);
}
$existingBooking = Booking::query()
    ->where('user_id', $user->getKey())
    ->where('class_id', $class->getKey())
    ->where('status', 'confirmed')
    ->first();
if ($existingBooking) {
    return redirect()
        ->route('dashboard.bookings')
        ->with('status', __('site.dashboard.bookings.already_booked'));}
```

Η ροή ξεκινά με έλεγχο ύπαρξης των πινάκων bookings και classes, ώστε το σύστημα να αποφεύγει σφάλματα σε μη ολοκληρωμένο περιβάλλον. Ακολουθεί validation για το class_id (πρέπει να υπάρχει στη βάση) και προαιρετικά για recommendation_impression_id.

Έπειτα γίνεται ανάκτηση του μαθήματος μαζί με μετρητή επιβεβαιωμένων κρατήσεων. Ο controller απορρίπτει:

- Προπονήσεις/μαθήματα που έχουν ήδη ξεκινήσει,
- Προπονήσεις/μαθήματα πλήρη ως προς τη χωρητικότητα,
- Κρατήσεις που ο ίδιος χρήστης έχει ήδη κάνει.

Με αυτά τα βήματα διασφαλίζεται ότι κάθε νέα κράτηση είναι έγκυρη και “λογική”.

4.4.1.2 Δημιουργία κράτησης & attribution

```php
<?php
    $booking = Booking::query()->create([
        'user_id' => $user->getKey(),
        'class_id' => $class->getKey(),
        'booking_date' => $class->start_time,
        'status' => 'confirmed',
    ]);
    $this->attachBookingAttribution(
        $booking,
        isset($validated['recommendation_impression_id'])
            ? (int) $validated['recommendation_impression_id']
            : null,    );
    return redirect()
        ->route('dashboard.bookings')
        ->with('status', __('site.dashboard.bookings.created_success'));
```

Αφού περάσουν οι έλεγχοι, δημιουργείται η κράτηση με status = confirmed. Στη συνέχεια, αν υπάρχει recommendation_impression_id, γίνεται σύνδεση της κράτησης με την αντίστοιχη πρόταση μέσω attachBookingAttribution. Αυτό επιτρέπει να γνωρίζουμε αν μια κράτηση προήλθε από recommendation, άρα είναι κρίσιμο για την αξιολόγηση του module προτάσεων.

4.4.2 Έλεγχοι ακύρωσης και αλλαγή κατάστασης

Ακολουθεί η ροή ακύρωσης, όπου εφαρμόζονται έλεγχοι ασφάλειας και χρονικοί περιορισμοί

```php
<?php
  public function cancelBooking(Request $request, Booking $booking): RedirectResponse
  {
    if (! Schema::hasTable('bookings')) {
        return redirect()
            ->route('dashboard.bookings')
            ->with('status', __('site.dashboard.bookings.unavailable'));
    }
    if ((int) $booking->user_id !== (int) $request->user()->getKey()) {
        abort(403);
    }
    if ($booking->status === 'cancelled') {
        return redirect()
            ->route('dashboard.bookings')
            ->with('status', __('site.dashboard.bookings.already_cancelled'));
```

```
        }
        if (! $booking->booking_date || ! $booking->booking_date->isFuture()) {
            return redirect()
                ->route('dashboard.bookings')
                ->with('status', __('site.dashboard.bookings.cannot_cancel_past'));
        }
        $booking->status = 'cancelled';
        $booking->save();
        return redirect()
            ->route('dashboard.bookings')
            ->with('status', __('site.dashboard.bookings.cancelled_success'));
    }
```

Η ακύρωση επιτρέπεται μόνο αν: (α) ο πίνακας υπάρχει, (β) η κράτηση ανήκει στον χρήστη, (γ) δεν έχει ήδη ακυρωθεί, και (δ) αφορά μελλοντικό μάθημα. Έτσι αποφεύγεται η αλλοίωση ιστορικού και προστατεύονται οι κρατήσεις άλλων χρηστών. Αν όλα είναι έγκυρα, γίνεται απλή αλλαγή κατάστασης σε cancelled και αποθήκευση.

4.4.3 Ροή συνδρομής / Δημιουργία και συγχρονισμός

Στη συνέχεια παρουσιάζεται η δημιουργία συνδρομής μέσω Stripe/Cashier και η ενημέρωση της τοπικής βάσης.

4.4.3.1 Δημιουργία συνδρομής μέσω Cashier

```
<?php
    public function store(Request $request): RedirectResponse
    {
        if (! Schema::hasTable('membership_types')) {
            return back()->with('status', __('site.dashboard.subscriptions.membership_types_missing'));
        }

        $validated = $request->validate([
            'membership_type_id' => ['required', 'integer', 'exists:membership_types,id'],
            'payment_method' => ['required', 'string'],
        ]);

        $user = $request->user();
        $membershipType = MembershipType::query()->findOrFail($validated['membership_type_id']);

        if (empty($membershipType->stripe_price_id) || str_starts_with($membershipType->stripe_price_id,
'price_placeholder_')) {
            return back()->with('status', __('site.dashboard.subscriptions.missing_price_id'));
```

```
    }
    try {
        $user->newSubscription('default', $membershipType->stripe_price_id)
            ->create($validated['payment_method']);
    } catch (IncompletePayment $exception) {
        return redirect()->route('cashier.payment', [
            $exception->payment->id,
            'redirect' => route('dashboard.subscriptions'),
        ]);
    }
```

Ο χρήστης επιλέγει πακέτο (membership_type_id) και μέθοδο πληρωμής. Γίνεται έλεγχος ότι το πακέτο έχει έγκυρο stripe_price_id, αλλιώς η αγορά απορρίπτεται. Η κλήση newSubscription(...)->create(...) είναι το σημείο όπου το Cashier δημιουργεί πραγματική συνδρομή στο Stripe. Αν η πληρωμή είναι "incomplete", ο χρήστης ανακατευθύνεται στο ειδικό flow του Cashier για ολοκλήρωση (3-D secure κ.λπ.).

4.4.3.2 Ενημέρωση τοπικής βάσης και μέλους

```
<?php
    $subscription = $user->subscription('default');
    $durationMonths = max(1, (int) $membershipType->duration_months);
    $membershipStartDate = now()->toDateString();
    $membershipEndDate = now()->addMonths($durationMonths)->toDateString();
    if ($subscription && Schema::hasTable('subscriptions')) {
        $subscription->forceFill([
            'start_date' => $membershipStartDate,
            'end_date' => $membershipEndDate,
            'price' => $membershipType->price,
            'status' => 'active',
        ])->save();
    }
    if (Schema::hasTable('members')) {
        $member = Member::query()->firstOrCreate(
            ['user_id' => $user->getKey()],
            [
                'name' => $user->name,
                'email' => $user->email,
                'phone' => $user->phone,
                'address' => $user->address,
            ]
```

```
        );
        $member->membership_type_id = $membershipType->getKey();
        $member->membership_start_date = $membershipStartDate;
        $member->membership_end_date = $membershipEndDate;
        $member->save();
    }
```

Μετά τη δημιουργία στο Stripe, ενημερώνεται η τοπική βάση με ημερομηνίες συνδρομής, τιμή και κατάσταση. Παράλληλα ενημερώνεται/δημιουργείται το member προφίλ ώστε να είναι συνεπές με τη νέα συνδρομή. Έτσι το σύστημα διαθέτει τοπικό state και δεν εξαρτάται από ζωντανές κλήσεις στο Stripe για κάθε προβολή.

4.4.3.3 Καταγραφή πληρωμής

```
<?php
    if (Schema::hasTable('payments')) {
        Payment::query()->create([
            'subscription_id' => $subscription?->getKey(),
            'user_id' => $user->getKey(),
            'member_id' => optional($user->member)->getKey(),
            'amount' => $membershipType->price,
            'payment_date' => now()->toDateString(),
            'method' => 'online',
            'payment_method' => 'stripe',
            'transaction_id' => $subscription?->stripe_id,
            'status' => 'completed',
        ]);
    }
```

Η πληρωμή αποθηκεύεται ξεχωριστά στον πίνακα payments ως οικονομικό ιστορικό. Η χρήση transaction_id με stripe_id επιτρέπει αντιστοίχιση της τοπικής πληρωμής με τη συναλλαγή στο Stripe. Έτσι, ακόμη κι αν το Stripe είναι εξωτερικό σύστημα, η εφαρμογή διατηρεί πλήρη, τοπική εικόνα.

4.4.4 Διαχείριση συνδρομής

Τέλος, καλύπτεται η διαχείριση της συνδρομής (ακύρωση, επαναφορά και πρόσβαση στο billing portal).

```
<?php
  public function destroy(Request $request): RedirectResponse
  {
    $subscription = $request->user()->subscription('default');
    if (! $subscription || ! $subscription->valid()) {
      return back()->with('status', __('site.dashboard.subscriptions.no_active_subscription'));
```

```
        }
        $subscription->cancel();
        return to_route('dashboard.subscriptions')
            ->with('status', __('site.dashboard.subscriptions.cancel_success'));
    }
    public function resume(Request $request): RedirectResponse
    {
        $subscription = $request->user()->subscription('default');
        if (! $subscription || ! $subscription->onGracePeriod()) {
            return back()->with('status', __('site.dashboard.subscriptions.not_on_grace_period'));
        }
        $subscription->resume();
        return to_route('dashboard.subscriptions')
            ->with('status', __('site.dashboard.subscriptions.resume_success'));
    }
    public function billingPortal(Request $request): RedirectResponse
    {
        return $request->user()->redirectToBillingPortal(route('dashboard.subscriptions'));
    }
```

Η ακύρωση απαιτεί ενεργή συνδρομή και ενημερώνει το Stripe μέσω Cashier. Η επαναφορά επιτρέπεται μόνο σε συνδρομές που βρίσκονται σε grace period, αποτρέποντας λανθασμένες επαναφορές. Το billingPortal δίνει πρόσβαση στο επίσημο portal του Stripe για αλλαγές πληρωμών ή στοιχείων, μειώνοντας την ανάγκη για custom UI.

4.5 Ασφάλεια, έλεγχος πρόσβασης και πολιτικές

Η ασφάλεια της πλατφόρμας βασίζεται σε τρία επίπεδα: (α) μηχανισμούς αυθεντικοποίησης, (β) middleware που περιορίζουν την πρόσβαση σε προστατευμένες περιοχές, και (γ) πολιτικές (policies) που εφαρμόζουν λεπτομερή κανόνες ανά μοντέλο. Τα αποσπάσματα που ακολουθούν δείχνουν πώς αυτά τα επίπεδα συνεργάζονται για να προστατεύουν δεδομένα και λειτουργίες.

4.5.1 Αυθεντικοποίηση (ροές του Laravel breeze)

```
<?php
Route::middleware('guest')->group(function (): void {
    Route::get('/register', [RegisteredUserController::class, 'create'])
        ->name('register');
    Route::post('/register', [RegisteredUserController::class, 'store']);
    Route::get('/login', [AuthenticatedSessionController::class, 'create'])
        ->name('login');
```

```
    Route::post('/login', [AuthenticatedSessionController::class, 'store']);
    Route::get('/forgot-password', [PasswordResetLinkController::class, 'create'])
        ->name('password.request');
    Route::post('/forgot-password', [PasswordResetLinkController::class, 'store'])
        ->name('password.email');
    Route::get('/reset-password/{token}', [NewPasswordController::class, 'create'])
        ->name('password.reset');
    Route::post('/reset-password', [NewPasswordController::class, 'store'])
        ->name('password.store');
});
Route::middleware('auth')->group(function (): void {
    Route::get('/verify-email', EmailVerificationPromptController::class)
        ->name('verification.notice');
    Route::get('/verify-email/{id}/{hash}', VerifyEmailController::class)
        ->middleware(['signed', 'throttle:6,1'])
        ->name('verification.verify');
    Route::post('/email/verification-notification', [EmailVerificationNotificationController::class, 'store'])
        ->middleware('throttle:6,1')
        ->name('verification.send');
    Route::get('/confirm-password', [ConfirmablePasswordController::class, 'show'])
        ->name('password.confirm');
    Route::post('/confirm-password', [ConfirmablePasswordController::class, 'store']);
    Route::get('/profile', [ProfileController::class, 'edit'])
        ->name('profile.edit');
    Route::patch('/profile', [ProfileController::class, 'update'])
        ->name('profile.update');
    Route::delete('/profile', [ProfileController::class, 'destroy'])
        ->name('profile.destroy');
    Route::put('/password', [PasswordController::class, 'update'])
        ->name('password.update');
    Route::post('/logout', [AuthenticatedSessionController::class, 'destroy'])
        ->name('logout');
});
```

Οι παραπάνω ροές προέρχονται από το Laravel Breeze, το οποίο αποτελεί ένα “starter kit” αυθεντικοποίησης για Laravel. Το Breeze παρέχει έτοιμα endpoints, controllers και views για βασικές λειτουργίες (register, login, password reset, email verification), ώστε να μην χρειάζεται να υλοποιηθούν από την αρχή. Στο έργο αξιοποιείται ως βάση για την ασφάλεια του χρήστη, ενώ η υπόλοιπη επιχειρησιακή λογική (κρατήσεις, συνδρομές κ.λπ.) αναπτύσσεται πάνω σε αυτή την έτοιμη υποδομή

Οι ροές χωρίζονται σε δύο ομάδες: guest για μη αυθεντικοποιημένους χρήστες (register/login/reset) και auth για ήδη συνδεδεμένους (profile, password, logout). Έτσι αποφεύγεται πρόσβαση σε λάθος ροές, π.χ. να γίνει reset password από ήδη logged-in χρήστη. Επιπλέον, η επιβεβαίωση email προστατεύεται με signed και throttle, ώστε να μειώνονται κακόβουλες προσπάθειες.

4.5.2 Middleware προστασίας dashboard

Η επόμενη γραμμή άμυνας είναι ο έλεγχος πρόσβασης στο dashboard, όπου βρίσκονται οι βασικές λειτουργίες του χρήστη.

```
<?php
   public function handle(Request $request, Closure $next): Response
   {
      if (! Auth::check()) {
         return redirect()->route('filament.admin.auth.login');
      }
      return $next($request);
   }
```

Το middleware ελέγχει αν ο χρήστης είναι αυθεντικοποιημένος. Αν όχι, γίνεται redirect στη σελίδα login. Έτσι διασφαλίζεται ότι όλες οι dashboard λειτουργίες προστατεύονται κεντρικά, χωρίς να απαιτούνται μεμονωμένοι έλεγχοι μέσα στους controllers.

4.5.3 Πολιτική πληρωμών

```
<?php
   public function view(User $user, Payment $payment): bool
   {
      if (method_exists($user, 'isAdmin') && $user->isAdmin()) {
         return true;
      }
      return $payment->user_id === $user->getKey();
   }
   public function update(User $user, Payment $payment): bool
   {
      if (method_exists($user, 'isAdmin') && $user->isAdmin()) {
         return true;
      }
      return $payment->user_id === $user->getKey();
   }
   public function delete(User $user, Payment $payment): bool
   {
      if (method_exists($user, 'isAdmin') && $user->isAdmin()) {
```

```
        return true;    }
    return $payment->user_id === $user->getKey();
}
public function create(User $user): bool
{
    return method_exists($user, 'isAdmin') && $user->isAdmin();
}
```

Οι διαχειριστές έχουν πλήρη πρόσβαση, ενώ οι απλοί χρήστες μπορούν να δουν/ενημερώσουν/διαγράψουν μόνο πληρωμές που τους ανήκουν. Προσθέτετε επίσης η δυνατότητα για την καταχώρηση χειροκίνητης πληρωμής, για την πιθανότητα της πληρωμής εκτός συστήματος και ο διαχειριστής να θέλει να την καταγράψει. Η δημιουργία νέας πληρωμής επιτρέπεται μόνο σε διαχειριστή, ώστε να αποφεύγονται χειροκίνητες καταχωρίσεις από μη εξουσιοδοτημένους χρήστες. Με αυτόν τον τρόπο, τα οικονομικά δεδομένα προστατεύονται αυστηρά.

4.5.4 Κεντρική χαρτογράφηση πολιτικών

Τέλος, οι πολιτικές πρέπει να δηλωθούν κεντρικά ώστε το Laravel να γνωρίζει ποια policy αντιστοιχεί σε κάθε μοντέλο.

```
<?php
  protected $policies = [
      Payment::class => PaymentPolicy::class,
      User::class => UserPolicy::class,
      GymClass::class => GymClassPolicy::class,
      Booking::class => BookingPolicy::class,
  ];
  public function boot(): void
  {
      $this->registerPolicies();
  }
```

Η χαρτογράφηση Model → Policy ενεργοποιεί επίσημα τους κανόνες πρόσβασης. Από εδώ και πέρα, κάθε έλεγχος εξουσιοδότησης (can, authorize) γνωρίζει ποια policy πρέπει να χρησιμοποιήσει. Έτσι εξασφαλίζεται ενιαία και συνεπής εφαρμογή των κανόνων σε όλο το σύστημα.

4.6 Επίλογος κεφαλαίου

Το κεφάλαιο αυτό παρουσίασε τη δομή και τη λογική του συστήματος από αρχιτεκτονική σκοπιά: τις βασικές οντότητες της βάσης, τις κύριες ροές (κρατήσεις, συνδρομές, πληρωμές) και τους μηχανισμούς ασφάλειας/εξουσιοδότησης. Με αυτό το υπόβαθρο, τα επόμενα κεφάλαια μπορούν να εστιάσουν στην πρακτική υλοποίηση και στη διεπαφή χρήστη, χωρίς να χάνεται η συνοχή με την υποκείμενη αρχιτεκτονική.

Κεφάλαιο 5ο: Υλοποίηση Πλατφόρμας

Το κεφάλαιο αυτό επικεντρώνεται στην πρακτική υλοποίηση της πλατφόρμας, παρουσιάζοντας τις βασικές λειτουργίες όπως αυτές αποτυπώνονται στον κώδικα και στη διεπαφή χρήστη. Κάθε υποενότητα συνοδεύεται από επιλεγμένα αποσπάσματα (με μικρά screenshots) και αναλυτική επεξήγηση, ώστε να φαίνεται καθαρά πώς συνδέονται οι απαιτήσεις του συστήματος με την τελική εμπειρία χρήσης.

5.1 Δημόσιο site (σελίδες/πλοήγηση)

Το δημόσιο site αποτελεί το "βιτρίνα" της πλατφόρμας και περιλαμβάνει τις βασικές ενημερωτικές σελίδες: αρχική, προπονήσεις, trainers, πακέτα και επικοινωνία. Κάθε σελίδα υλοποιείται με ξεχωριστή μέθοδο στον PublicPageController και αντίστοιχο Blade view, ενώ τα δεδομένα που εμφανίζονται είναι σαφώς διαχωρισμένα από τις προστατευμένες λειτουργίες του dashboard.

Στην αρχή κάθε view εμφανίζεται η δήλωση @section('title', __('site.nav.contact')). Η οδηγία @section ορίζει το περιεχόμενο της περιοχής title στο κύριο layout, ώστε κάθε σελίδα να έχει σωστό τίτλο. Η συνάρτηση __() αντλεί το αντίστοιχο κείμενο από τα αρχεία μετάφρασης, χρησιμοποιώντας το key (π.χ. site.nav.contact). Έτσι ο τίτλος μπορεί να αλλάζει γλώσσα δυναμικά χωρίς να αλλάξει ο κώδικας της σελίδας.

5.1.1 Αρχική σελίδα

Ξεκινάμε από την αρχική σελίδα, η οποία λειτουργεί ως το βασικό σημείο πρώτης εντύπωσης και προβολής στατιστικών

```
<?php
   public function home(): View
   {
      $activeMembers = Schema::hasTable('members')
         ? Member::whereNotNull('membership_end_date', 'and')
            ->where('membership_end_date', '>=', now()->toDateString(), 'and')
            ->count('*')
         : (Schema::hasTable('users') && Schema::hasColumn('users', 'role')
            ? User::where('role', '=', 'client', 'and')->count('*')
            : 0);
      $weeklyClasses = Schema::hasTable('classes')
         ? GymClass::where('start_time', '>=', now()->startOfWeek(), 'and')
            ->where('start_time', '<=', now()->endOfWeek(), 'and')
            ->count('*')
         : 0;
      $trainerCount = (Schema::hasTable('users') && Schema::hasColumn('users', 'role'))
         ? User::where('role', '=', 'trainer', 'and')->count('*')
```

```
        : 0;
    return view('pages.public.home', [
        'stats' => [
            'activeMembers' => $activeMembers,
            'weeklyClasses' => $weeklyClasses,
            'trainerCount' => $trainerCount,
        ],
    ]);
}
```

Η αρχική σελίδα εμφανίζει τρία βασικά στατιστικά, τους παρουσιάζουμε παρακάτω. Οι έλεγχοι Schema::hasTable() προστατεύουν από σφάλματα σε περιβάλλον χωρίς migrations. Τα δεδομένα υπολογίζονται απευθείας στη βάση (count), ώστε να μην φορτώνονται περιττές εγγραφές. Το αποτέλεσμα περνά στο view ως stats array, κρατώντας το UI “ελαφρύ” και σαφές.

```
<?php
    public function home(): View
    {
        $activeMembers = Schema::hasTable('members')
            ? Member::whereNotNull('membership_end_date', 'and')
                ->where('membership_end_date', '>=', now()->toDateString(), 'and')
                ->count('*')
            : (Schema::hasTable('users') && Schema::hasColumn('users', 'role')
                ? User::where('role', '=', 'client', 'and')->count('*')
                : 0);
        $weeklyClasses = Schema::hasTable('classes')
            ? GymClass::where('start_time', '>=', now()->startOfWeek(), 'and')
                ->where('start_time', '<=', now()->endOfWeek(), 'and')
                ->count('*')
            : 0;
        $trainerCount = (Schema::hasTable('users') && Schema::hasColumn('users', 'role'))
            ? User::where('role', '=', 'trainer', 'and')->count('*')
            : 0;
        return view('pages.public.home', [
            'stats' => [
                'activeMembers' => $activeMembers,
                'weeklyClasses' => $weeklyClasses,
                'trainerCount' => $trainerCount,
            ],  ]);}
```

```
<?php
  <section class="relative -mt-10 min-h-[80vh]">
    <div class="absolute inset-0">
      <img
        src="{{ asset('images/unsplash1.jpg') }}"
        alt="{{ __('site.public.home.gallery_alt_1') }}"
        class="h-full w-full object-cover"
        loading="eager"
        decoding="async"
      />
      <div class="absolute inset-0 bg-gradient-to-b from-black/40 via-slate-950/45 to-slate-950/70"></div>
    </div>
    <div class="ui-container relative z-10 flex min-h-[80vh] items-center py-16">
      <div class="max-w-2xl text-center md:text-left">
        <h1 class="text-4xl font-bold tracking-tight text-white sm:text-5xl lg:text-6xl">
          {{ __('site.public.home.title') }}
        </h1>
        <p class="mt-4 text-base text-slate-200 sm:text-lg">
          {{ __('site.public.home.subtitle') }}
        </p>
```

Η hero περιοχή χρησιμοποιεί εικόνα φόντου και gradient για οπτική ένταση, ενώ όλα τα κείμενα περνούν από translation keys (__()), ώστε να υποστηρίζεται πολυγλωσσία. Τα στατιστικά εμφανίζονται με reusable components (x-ui.stat-card), κάτι που κάνει το UI πιο συνεπές και εύκολα επεκτάσιμο.

```
<?php
  <x-ui.section class="relative mt-0.7 bg-gradient-to-b from-transparent to-[#020617] py-16">
    <div class="home-stats grid gap-8 sm:grid-cols-2 lg:grid-cols-3">
      <x-ui.stat-card
        :label="__('site.public.home.active_members')"
        :value="number_format($stats['activeMembers'])"
        :trend="__('site.public.home.active_members_trend')"
      />
      <x-ui.stat-card
        :label="__('site.public.home.weekly_classes')"
        :value="number_format($stats['weeklyClasses'])"
        :trend="__('site.public.home.weekly_classes_trend')"
      />
```

```
        <x-ui.stat-card
            :label="__('site.public.home.certified_trainers')"
            :value="number_format($stats['trainerCount'])"
            :trend="__('site.public.home.certified_trainers_trend')"
        />
    </div>
</x-ui.section>
```

5.1.2 Σελίδα μαθημάτων

Εδώ παρουσιάζεται η δημόσια λίστα μαθημάτων και ο τρόπος προβολής τους στο UI

```
<?php
    public function classes(): View
    {
        $gymClasses = Schema::hasTable('classes')
            ? GymClass::query()
                ->with('trainer:id,name')
                ->orderBy('start_time')
                ->get()
            : Collection::make();

        return view('pages.public.classes', [
            'gymClasses' => $gymClasses,
        ]);
    }
```

Η δημόσια λίστα μαθημάτων φορτώνει όλα τα classes με trainer (μόνο id και name για οικονομία). Η ταξινόμηση γίνεται με βάση την ώρα έναρξης, ώστε το πρόγραμμα να έχει φυσική ροή.

```
<?php
        @if ($gymClasses->isEmpty())
            <x-ui.empty-state:title="__('site.public.classes.empty_title')"
:description="__('site.public.classes.empty_description')" />
        @else
            <div class="grid gap-6 md:grid-cols-2 lg:grid-cols-3">
                @foreach ($gymClasses as $gymClass)
                    <x-gym.class-card
                        :name="$gymClass->name"
                        :level="__('site.public.classes.capacity', ['count' => $gymClass->capacity])"
                        :duration="($gymClass->start_time && $gymClass->end_time) ? __('site.public.classes.duration_minutes',
['minutes' => $gymClass->start_time->diffInMinutes($gymClass->end_time)]) : __('site.public.classes.duration_tbd')"
```

```
                :description="$gymClass->description ?: __('site.public.classes.coach', ['name' => $gymClass->trainer?->name ?? 'TBD'])"
            />
        @endforeach
    </div>
@endif
```

Η προβολή χρησιμοποιεί empty-state όταν δεν υπάρχουν προπονήσεις/μαθήματα. Κάθε μάθημα αποδίδεται από component, με υπολογισμό διάρκειας σε λεπτά και fallback κείμενο όταν λείπει περιγραφή. Έτσι το UI παραμένει πλήρες ακόμα και όταν κάποια δεδομένα είναι κενά.

5.1.3 Σελίδα προπονητών

Η ενότητα καλύπτει την προβολή των γυμναστών και τα βασικά στοιχεία επικοινωνίας τους.

```
<?php
  public function trainers(): View
  {
    $trainers = (Schema::hasTable('users') && Schema::hasColumn('users', 'role'))
      ? User::query()
        ->where('role', 'trainer')
        ->orderBy('name', 'asc')
        ->get(['id', 'name', 'email', 'phone'])
      : Collection::make();
    return view('pages.public.trainers', [
      'trainers' => $trainers,
    ]);
  }
```

Η σελίδα προβάλλει μόνο χρήστες με ρόλο trainer, ενώ γίνεται επιλεκτική φόρτωση πεδίων (id, name, email, phone) ώστε να περιοριστεί το payload.

```
<?php
    @if ($trainers->isEmpty())
      <x-ui.empty-state:title="__('site.public.trainers.empty_title')" :description="__('site.public.trainers.empty_description')" />
    @else
      <div class="grid gap-6 md:grid-cols-2 lg:grid-cols-3">
        @foreach ($trainers as $trainer)
          <x-gym.trainer-card
            :name="$trainer->name"
            :specialty="__('site.public.trainers.role')"
            :bio="__('site.public.trainers.contact', ['value' => $trainer->phone ?: $trainer->email])"    />
        @endforeach
```

```
        </div>
    @endif
```

Η προβολή εμφανίζει κάρτες trainers με βασική πληροφορία και fallback (τηλέφωνο ή email). Η χρήση components κρατά συνεπές το layout και επιτρέπει εύκολη αισθητική προσαρμογή.

5.1.4 Σελίδα πλάνων συνδρομής

Παρουσιάζονται τα διαθέσιμα συνδρομητικά πακέτα με έμφαση στη σύγκριση και στην κλήση για εγγραφή.

```
<?php
  public function plans(): View
  {
    $membershipTypes = class_exists(MembershipType::class) && Schema::hasTable('membership_types')
      ? MembershipType::query()
        ->orderBy('price', 'asc')
        ->get(['id', 'name', 'description', 'price', 'duration_months'])
        ->map(function (MembershipType $plan) {
          [$displayName, $displayDescription] = $this->getTranslatedPlanText($plan->name, $plan->description);

          $plan->setAttribute('display_name', $displayName);
          $plan->setAttribute('display_description', $displayDescription);

          return $plan;
        })
      : Collection::make();

    return view('pages.public.plans', [
      'membershipTypes' => $membershipTypes,
    ]); }
```

Τα πακέτα φορτώνονται ταξινομημένα με τιμή, και μετά μετασχηματίζονται ώστε να έχουν display_name και display_description. Αυτό επιτρέπει μετάφραση χωρίς να “πειράζονται” οι αρχικές τιμές, διατηρώντας καθαρά τα δεδομένα της βάσης.

```
@section('content')
  <x-ui.page-header :title="__('site.public.plans.title')" :subtitle="__('site.public.plans.subtitle')">
    <x-slot name="actions">
      @guest
        <x-ui.button tag="a" href="{{ route('login') }}">
          {{ __('site.public.plans.cta_sign_in') }}
        </x-ui.button>
```

```
        @else
          <x-ui.button tag="a" href="{{ route('dashboard.subscriptions') }}">
            {{ __('site.public.plans.cta_subscribe') }}
          </x-ui.button>
        @endguest
      </x-slot>
    </x-ui.page-header>
```

Αρχικά, το πρώτο πράγμα που ελέγχουμε στην προβολή των πλάνων συνδρομής είναι ο ρόλος χρήστη. Στην περίπτωση που δεν είναι συνδεδεμένος με τον λογαριασμό του, εμφανίζεται ένα call-to-action (CTA) κουμπί που τον προτρέπει να συνδεθεί στον λογαριασμό του και να εγγραφεί στο γυμναστήριο. Σε διαφορετική περίπτωση είναι ένα CTA κουμπί που προτρέπει τον χρήστη στην αγορά συνδρομής και τον οδηγεί στη σελίδα με την δυνατότητα αγοράς συνδρομής.

```
<?php
    @php
        $featuredPlanId = $membershipTypes->sortByDesc('duration_months')->first()?->id;
    @endphp
```

```
<?php
        @if ($membershipTypes->isEmpty())
            <x-ui.empty-state:title="__('site.public.plans.empty_title')"
:description="__('site.public.plans.empty_description')" />
        @else
            <div class="grid gap-6 md:grid-cols-2 lg:grid-cols-3">
                @foreach ($membershipTypes as $plan)
                    <x-ui.card
                        @class([
                            'group relative overflow-hidden space-y-5 border transition duration-300 hover:-translate-y-1
hover:shadow-2xl',
                            'border-amber-300/40 bg-gradient-to-br from-amber-500/12 via-slate-900/85 to-slate-900/95
hover:shadow-amber-500/15' => (int) $plan->id === (int) $featuredPlanId,
                            'border-white/10 bg-slate-900/75 hover:border-amber-300/30 hover:shadow-black/40' => (int)
$plan->id !== (int) $featuredPlanId,
                        ]) >
```

Το featuredPlanId επιλέγει το πακέτο με τη μεγαλύτερη διάρκεια ώστε να εμφανιστεί ως “προτεινόμενο”. Στο view, οι κάρτες διαφοροποιούνται οπτικά με conditional classes. Έτσι γίνεται καθαρή έμφαση σε ένα πακέτο χωρίς να απαιτείται επιπλέον λογική στη βάση.

5.1.5 Σελίδα επικοινωνίας

Κλείνουμε με τη φόρμα επικοινωνίας, η οποία ολοκληρώνει το δημόσιο περιεχόμενο της πλατφόρμας.

```
<?php
  <x-ui.section>
    <x-ui.card>
      <form class="grid gap-4 md:grid-cols-2">
        <div>
          <label class="mb-1 block text-sm font-semibold text-slate-200">{{ __('site.public.contact.full_name') }}</label>
          <input type="text" class="w-full rounded-lg border-white/10 bg-slate-950/60 text-slate-100 focus:border-amber-400 focus:ring-amber-400/30" placeholder="{{ __('site.public.contact.name_placeholder') }}" />
        </div>
        <div>
          <label class="mb-1 block text-sm font-semibold text-slate-200">{{ __('site.public.contact.email') }}</label>
          <input type="email" class="w-full rounded-lg border-white/10 bg-slate-950/60 text-slate-100 focus:border-amber-400 focus:ring-amber-400/30" placeholder="{{ __('site.public.contact.email_placeholder') }}" />
        </div>
        <div class="md:col-span-2">
          <label class="mb-1 block text-sm font-semibold text-slate-200">{{ __('site.public.contact.message') }}</label>
          <textarea rows="5" class="w-full rounded-lg border-white/10 bg-slate-950/60 text-slate-100 focus:border-amber-400 focus:ring-amber-400/30" placeholder="{{ __('site.public.contact.message_placeholder') }}"></textarea>
        </div>
        <div class="md:col-span-2">
          <x-ui.button type="button">{{ __('site.public.contact.send') }}</x-ui.button>
        </div>
      </form>
    </x-ui.card>
  </x-ui.section>
```

Η φόρμα είναι στατική (button type="button"), λειτουργώντας ως UI παρουσίαση. Όλα τα labels/placeholder είναι μεταφρασμένα μέσω keys, διατηρώντας συνέπεια με το υπόλοιπο site. Η διάταξη σε grid επιτρέπει responsive μορφή χωρίς πρόσθετη JS λογική.

5.2 Dashboard χρήστη

Το dashboard είναι ο βασικός χώρος αλληλεπίδρασης του συνδεδεμένου χρήστη. Εδώ συγκεντρώνονται οι κρατήσεις, το προφίλ, η συνδρομή και οι προτάσεις, ώστε ο χρήστης να έχει μια ενιαία εικόνα και να μπορεί να εκτελεί τις κύριες ενέργειες χωρίς να μεταβαίνει σε διαφορετικά υποσυστήματα.

5.2.1 Επισκόπηση (dashboard home)

Η αρχική οθόνη του dashboard δίνει συνοπτικά στατιστικά, προτάσεις και πρόσφατη δραστηριότητα.

```
<div class="grid gap-6 md:grid-cols-3">
  <x-ui.stat-card :label="__('site.dashboard.index.upcoming_bookings')" :value="$stats['upcomingBookings']" :trend="__('site.dashboard.index.upcoming_bookings_trend')" />
```

```
    <x-ui.stat-card :label="__('site.dashboard.index.membership_plan')" :value="$stats['membershipPlan'] ?:
__('site.dashboard.index.no_active_plan')" :trend="__('site.dashboard.index.membership_plan_trend')" />

    <x-ui.stat-card :label="__('site.dashboard.index.attendance')" :value="$stats['attendanceThisMonth']"
:trend="__('site.dashboard.index.attendance_trend')" />

</div>
```

Οι κάρτες στατιστικών συγκεντρώνουν τρία βασικά στοιχεία: επερχόμενες κρατήσεις, τρέχον πρόγραμμα συνδρομής και παρουσίες του μήνα. Έτσι ο χρήστης έχει άμεση εικόνα της κατάστασής του χωρίς να πλοηγηθεί σε άλλες σελίδες. Τα labels είναι μεταφρασμένα μέσω keys, ενώ το membershipPlan έχει fallback ώστε να εμφανίζεται σαφές μήνυμα όταν δεν υπάρχει ενεργό πακέτο.

```
@foreach ($recommendations as $recommendation)
    @php
        $class = $recommendation['gymClass'];
        $duration = ($class->start_time && $class->end_time)
            ? __('site.public.classes.duration_minutes', ['minutes' => $class->start_time->diffInMinutes($class->end_time)])
            : __('site.public.classes.duration_tbd');
        $reasonLabel = __('site.dashboard.index.reasons.' . $recommendation['reason_key']);
    @endphp

    <x-ui.card class="space-y-3">
        <p class="text-lg font-semibold text-white">{{ $class->name }}</p>

        <p class="rounded-lg border border-amber-400/30 bg-amber-500/10 px-3 py-2 text-xs font-semibold text-amber-200">
            {{ __('site.dashboard.index.recommendation_reason') }}: {{ $reasonLabel }}
        </p>

        <form method="POST" action="{{ route('dashboard.bookings.store') }}">
            @csrf
            <input type="hidden" name="class_id" value="{{ $class->id }}">
            @if (! empty($recommendation['impression_id']))
                <input type="hidden" name="recommendation_impression_id" value="{{ $recommendation['impression_id'] }}">
            @endif
            <x-ui.button type="submit" class="w-full justify-center" data-rec-click="{{ $recommendation['impression_id'] ?? ''
}}">
                {{ __('site.dashboard.index.recommendation_cta') }}
            </x-ui.button>
        </form>
    </x-ui.card>
@endforeach
```

Κάθε πρόταση αποδίδεται ως κάρτα με διάρκεια, λόγο πρότασης και CTA για άμεση κράτηση. Το recommendation_impression_id περνά ως hidden πεδίο ώστε η κράτηση να μπορεί να συσχετιστεί με την πρόταση που την προκάλεσε. Έτσι το UI συνδέεται με το tracking χωρίς επιπλέον βήματα για τον χρήστη.

```
<script>
  (() => {
    const clickUrl = @json(route('dashboard.recommendations.click'));
    const csrfToken = @json(csrf_token());
    const links = document.querySelectorAll('[data-rec-click]');
    if (! links.length) {
      return;
    }
    links.forEach((link) => {
      link.addEventListener('click', () => {
        const impressionId = link.getAttribute('data-rec-click');
        if (! impressionId) {
          return;
        }
        const payload = new FormData();
        payload.append('_token', csrfToken);
        payload.append('impression_id', impressionId);
        if (navigator.sendBeacon) {
          navigator.sendBeacon(clickUrl, payload);
          return;
        }
        fetch(clickUrl, {
          method: 'POST',
          headers: {
            'X-CSRF-TOKEN': csrfToken,
            'Content-Type': 'application/json',
          },
          body: JSON.stringify({ impression_id: impressionId }),
          keepalive: true,
        });
      });
    });
  })();
</script>
```

Το script καταγράφει κάθε click σε προτεινόμενο μάθημα. Η χρήση sendBeacon επιτρέπει την αποστολή δεδομένων ακόμη και όταν ο χρήστης μεταφέρεται σε άλλη σελίδα (π.χ. μετά την κράτηση). Αν δεν υποστηρίζεται, γίνεται fallback σε fetch. Έτσι το tracking παραμένει αξιόπιστο χωρίς να "κόβει" την εμπειρία χρήστη.

5.2.2 Σελίδα κρατήσεων

Η ενότητα αυτή δείχνει τις διαθέσιμες προπονήσεις και το ιστορικό/κατάσταση των κρατήσεων

```
@foreach ($availableClasses as $class)
  <x-ui.card class="space-y-3">
    <p class="text-lg font-semibold text-white">{{ $class->name }}</p>

    <p class="text-sm text-slate-300">{{ $class->description ?: __('site.dashboard.bookings.table.class_fallback') }}</p>

    <form method="POST" action="{{ route('dashboard.bookings.store') }}">
      @csrf
      <input type="hidden" name="class_id" value="{{ $class->id }}">
      <x-ui.button type="submit" class="w-full justify-center">
        {{ __('site.dashboard.bookings.book_now') }}
      </x-ui.button>
    </form>
  </x-ui.card>
@endforeach
```

Ο χρήστης βλέπει διαθέσιμες προπονήσεις/μαθήματα και μπορεί να κάνει κράτηση με ένα κλικ. Το class_id περνά σε hidden πεδίο, ενώ το @csrf εξασφαλίζει προστασία από CSRF επιθέσεις.

```
@foreach ($bookings as $booking)
  <x-ui.table.row>
    <x-ui.table.cell>{{ $booking->gymClass?->name ?? __('site.dashboard.bookings.table.class_fallback') }}</x-ui.table.cell>
    <x-ui.table.cell>{{ $booking->booking_date?->format('M j, Y g:i A') }}</x-ui.table.cell>
    <x-ui.table.cell class="{{ ($booking->status ?? 'confirmed') === 'confirmed' ? 'text-emerald-300' : 'text-amber-300' }}">
      {{ ucfirst($booking->status ?? 'confirmed') }}
    </x-ui.table.cell>
    <x-ui.table.cell>
      @if (($booking->status ?? 'confirmed') === 'confirmed' && $booking->booking_date?->isFuture())
        <form method="POST" action="{{ route('dashboard.bookings.cancel', $booking) }}">
          @csrf
          <x-ui.button type="submit" size="sm" variant="secondary">
            {{ __('site.dashboard.bookings.cancel') }}
```

```
                </x-ui.button>
            </form>
        @else
            <span class="text-xs text-slate-400">-</span>
        @endif
    </x-ui.table.cell>
  </x-ui.table.row>
@endforeach
```

Οι κρατήσεις εμφανίζονται σε πίνακα με κατάσταση και δυνατότητα ακύρωσης μόνο για ενεργές και μελλοντικές κρατήσεις. Έτσι αποτρέπονται ακυρώσεις "ιστορικών" δεδομένων.

5.2.3 Προφίλ χρήστη

Η ενότητα προφίλ επιτρέπει στον χρήστη να ενημερώνει στοιχεία λογαριασμού και προσωπικές προτιμήσεις.

```
<form method="POST" action="{{ route('dashboard.profile.update') }}" class="space-y-6">
  @csrf
  @method('PUT')
  <x-ui.card>
    <h2 class="text-lg font-semibold text-white">{{ __('site.dashboard.profile.account_section_title') }}</h2>
    <div class="mt-5 grid gap-4 md:grid-cols-2">
      <div>
        <label class="mb-1 block text-sm font-semibold text-slate-200" for="email">{{ __('site.dashboard.profile.email') }}</label>
        <input id="email" name="email" type="email" value="{{ old('email', $user->email) }}" class="w-full rounded-lg border-white/10 bg-slate-950/60 text-slate-100 focus:border-amber-400 focus:ring-amber-400/30" required />
      </div>
    </div>
  </x-ui.card>
  <x-ui.card>
    <p class="mt-1 text-sm text-slate-300">{{ __('site.dashboard.profile.preferences_section_subtitle') }}</p>
    <div class="mt-5 grid gap-4 md:grid-cols-3">
      <div>
        <label class="mb-1 block text-sm font-semibold text-slate-200" for="primary_goal">{{ __('site.dashboard.profile.primary_goal') }}</label>
        <select id="primary_goal" name="primary_goal" class="w-full rounded-lg border-white/10 bg-slate-950/60 text-slate-100 focus:border-amber-400 focus:ring-amber-400/30">
          <option value="">{{ __('site.dashboard.profile.select_placeholder') }}</option>
          @foreach ($goalOptions as $value => $label)
            <option value="{{ $value }}" @selected(old('primary_goal', $member?->primary_goal) === $value)>{{ $label }}</option>
```

```
            @endforeach
          </select>
        </div>
      </div>
    </x-ui.card>
    <x-ui.button type="submit">{{ __('site.dashboard.profile.save') }}</x-ui.button>
</form>
```

Η φόρμα υποστηρίζει ενημέρωση λογαριασμού (email/κωδικός) και προτιμήσεων που χρησιμοποιούνται αργότερα για εξατομίκευση. Η χρήση old(...) διατηρεί τα στοιχεία σε περίπτωση validation errors, ενώ οι επιλογές προτιμήσεων είναι μεταφρασμένες για συνέπεια στη διεπαφή.

5.2.4 Συνδρομές

Τέλος, η σελίδα συνδρομών συγκεντρώνει πληροφορίες για το τρέχον πακέτο και παρέχει ενέργειες διαχείρισης.

```
<p class="text-sm uppercase tracking-wide text-slate-300">
{{ __('site.dashboard.subscriptions.current_plan') }}</p>
<p class="text-2xl font-bold text-white md:text-3xl">
{{ $currentPlanName ?? __('site.dashboard.subscriptions.no_active_plan') }}</p>
<p class="text-sm leading-relaxed text-slate-300">
{{ $currentPlanDescription ?? __('site.dashboard.subscriptions.plan_fallback') }}</p>
```

Η σελίδα εμφανίζει συνοπτικά το ενεργό πακέτο και περιγραφή του. Έτσι ο χρήστης γνωρίζει άμεσα σε ποιο πλάνο ανήκει χωρίς να ανατρέχει σε άλλες ενότητες.

```
<form id="subscription-form" method="POST" action="{{ route('dashboard.subscriptions.store') }}" class="space-y-4">
    @csrf
    <fieldset class="space-y-2 rounded-xl border border-white/10 bg-black/20 p-3">
      <legend class="mb-2 text-sm font-medium text-slate-200">{{ __('site.dashboard.subscriptions.plan_label') }}</legend>
      @foreach ($membershipTypes as $plan)
        <label class="relative block cursor-pointer" data-plan-option>
          <div class="rounded-lg border px-3 py-2 text-sm transition" data-plan-card>
            <input type="radio" name="membership_type_id" value="{{ $plan->id }}" class="mt-0.5 h-4 w-4 border-slate-
500 bg-slate-900 text-amber-400 focus:ring-amber-300/50">
            <p class="font-semibold text-slate-100">{{ $plan->display_name ?? $plan->name }}</p>
          </div>
        </label>
      @endforeach   </fieldset>
    <div>
      <label class="mb-2 block text-sm text-slate-200">{{ __('site.dashboard.subscriptions.card_details') }}</label>
      <div id="card-element" class="rounded-xl border border-slate-600/80 bg-slate-900/80 px-3 py-3"></div>
    </div>
</form>
```

Ο χρήστης επιλέγει πακέτο μέσω radio επιλογών και στη συνέχεια εμφανίζεται περιοχή για εισαγωγή στοιχείων κάρτας. Η πραγματική διαχείριση πληρωμής (Stripe Elements) αναλύεται στο 4.4, εδώ όμως φαίνεται η δομή της φόρμας και ο τρόπος επιλογής πακέτου.

5.3 Admin panel (Πάνελ διαχείρισης)«

Το Filament χρησιμοποιείται ως admin panel για τη διαχείριση βασικών πόρων (χρήστες, προπονήσεις, κρατήσεις, πληρωμές). Κάθε πόρος υλοποιείται ως Resource, όπου ορίζονται το μοντέλο, οι φόρμες, οι πίνακες και οι σελίδες. Επιπλέον, εφαρμόζεται περιορισμός πρόσβασης ανά ρόλο μέσα από helper μεθόδους και policies.

Στα αρχικά στάδια, ορισμένες λειτουργίες και προβολές ήταν διαθέσιμες μέσα από το admin panel, με περιορισμούς ώστε κάθε χρήστης να βλέπει μόνο τα δικά του δεδομένα. Ωστόσο, αυτό μεταφέρθηκε σταδιακά στο ειδικό dashboard χρήστη, ώστε η εμπειρία να είναι πιο καθαρή και προσανατολισμένη στον τελικό χρήστη. Έτσι το Filament παρέμεινε αποκλειστικά για διοικητική διαχείριση, ενώ το dashboard καλύπτει τις καθημερινές ανάγκες των μελών.

Στο Filament, το Resource είναι η βασική μονάδα διαχείρισης: αντιστοιχεί σε ένα Eloquent μοντέλο και "πακετάρει" όλη τη λειτουργικότητα CRUD για το συγκεκριμένο αντικείμενο. Μέσα στο Resource ορίζονται τα labels και η πλοήγηση, οι διαθέσιμες σελίδες (index, create, edit, view) και η σύνδεση με φόρμες και πίνακες. Έτσι, το admin UI δεν υλοποιείται χειροκίνητα, αλλά παράγεται συστηματικά από τις δηλώσεις του Resource.»

Επιπλέον, το Resource επιτρέπει ενσωμάτωση κανόνων πρόσβασης μέσω policies ή scopes στο query. Αυτό σημαίνει ότι η ίδια οντότητα μπορεί να παρουσιάζεται διαφορετικά ανά ρόλο, χωρίς να αλλάζει ο κώδικας των views. Η διάσπαση σε ξεχωριστές κλάσεις (π.χ. GymClassForm, GymClassesTable) κρατά την υλοποίηση καθαρή και επιτρέπει εύκολη επέκταση όταν προστίθενται νέα πεδία ή φίλτρα.

5.3.1 Φόρμες (Form) vs Πινάκων (Table)

Στο Filament, κάθε Resource χωρίζεται πρακτικά σε δύο βασικά κομμάτια: τη φόρμα (Form) για δημιουργία/επεξεργασία και τον πίνακα (Table) για λίστα/αναζήτηση. Έτσι ο κώδικας παραμένει καθαρός και τα responsibilities είναι ξεκάθαρα. Παρακάτω έχουμε ένα απόσπασμα κώδικα φόρμας και πίνακα

```
<?php
TextInput::make('name')
        ->label(__('filament-admin.fields.name'))
        ->required()
        ->maxLength(255),
Select::make('trainer_id')
        ->label(__('filament-admin.fields.trainer'))
        ->relationship(
                'trainer',
                'name'.
                fn ($query) => $query
                        ->wherein('role', ['admin','trainer'], 'and',
```

```
                    ->orderBy('name', 'asc')
        )
        ->searchable()
        ->preload()
        ->required(),
```

Η φόρμα δηλώνει τα πεδία που επεξεργάζεται ο διαχειριστής. Εδώ το trainer_id συνδέεται με σχέση (relationship) και φιλτράρει μόνο χρήστες με ρόλο admin ή trainer. Η επιλογή γίνεται searchable και προφορτωμένη, για καλύτερη εμπειρία. Αυτό δείχνει πώς η φόρμα δεν είναι απλό input, αλλά περιλαμβάνει business λογική επιλογής.

```
<?php
return $table->columns([
    TextColumn::make('name')->label(__('filament-admin.fields.name'))->searchable()->sortable(),
    TextColumn::make('start_time')->label(__('filament-admin.fields.start'))->dateTime()->sortable(),
    TextColumn::make('end_time')->label(__('filament-admin.fields.end'))->dateTime()->sortable(),
    TextColumn::make('trainer.name')->label(__('filament-admin.fields.trainer'))->sortable(),
])->filters([
    Filter::make('hide_expired')
        ->label(__('filament-admin.filters.hide_expired'))
        ->query(fn ($query) => $query->where('end_time', '>=', now(), 'and'))
        ->toggle(),
])->recordActions([
    ViewAction::make(),
    EditAction::make()->visible(fn ($record) => static::canEdit($record)),
    DeleteAction::make()->visible(fn ($record) => static::canDelete($record)),
]);
```

Ο πίνακας ορίζει πώς προβάλλονται τα δεδομένα: sortable columns, φίλτρα και actions. Το φίλτρο hide_expired κρύβει ληγμένα προπονήσεις/μαθήματα, ενώ τα actions Edit/Delete εμφανίζονται μόνο αν ο χρήστης έχει δικαιώματα. Έτσι ο πίνακας δεν είναι απλώς λίστα, αλλά "ενεργό" εργαλείο διαχείρισης.

```
<?php
TextColumn::make('status')
    ->label(__('filament-admin.fields.status'))
    ->formatStateUsing(fn (?string $state) => __('filament-admin.statuses.' . ($state ?? 'pending')))
    ->sortable(),
```

Το πεδίο status μεταφράζεται δυναμικά μέσω formatStateUsing, ώστε οι τιμές της βάσης να προβάλλονται ως κατανοητές, μεταφρασμένες ετικέτες στη διεπαφή. Αυτό δείχνει ότι το Table layer μπορεί να "μορφοποιεί" δεδομένα, όχι μόνο να τα εμφανίζει.

5.3.2 Resource μαθημάτων (gymClassResource)

Η υποενότητα δείχνει πώς οργανώνεται ένας βασικός πόρος στο Filament, με φόρμα, πίνακα και σελίδες.

```php
<?php
class GymClassResource extends Resource
{
    use ResourceAuthorization;
    protected static ?string $model = GymClass::class;
    protected static string|BackedEnum|null $navigationIcon = Heroicon::OutlinedCalendarDays;
    public static function getNavigationLabel(): string
    {
        return __('filament-admin.navigation.classes');
    }
    public static function form(Schema $schema): Schema
    {
        return GymClassForm::configure($schema);
    }
    public static function table(Table $table): Table
    {
        return GymClassesTable::configure($table);
    }
    public static function getPages(): array
    {
        return [
            'index' => ListGymClasses::route('/'),
            'create' => CreateGymClass::route('/create'),
            'view' => ViewGymClass::route('/{record}'),
            'edit' => EditGymClass::route('/{record}/edit'),
        ];
    }
}
```

Το GymClassResource συνδέει το admin panel με το μοντέλο GymClass. Οι μέθοδοι form() και table() παραπέμπουν σε ξεχωριστές κλάσεις διαμόρφωσης, κάτι που κρατά την υλοποίηση καθαρή και επεκτάσιμη. Η getPages() ορίζει τις βασικές σελίδες CRUD. Με αυτόν τον τρόπο, το Filament δημιουργεί αυτόματα τη διαχείριση μαθημάτων χωρίς να χρειάζεται χειροκίνητο UI.

5.3.3 Περιορισμός πρόσβασης σε resources

Η ενότητα δείχνει πώς το Filament περιορίζει τη διαθεσιμότητα ενός resource και εφαρμόζει επιπλέον έλεγχο στο query.

```
<?php
   public static function shouldRegisterNavigation(): bool
   {
      $user = static::currentUser();

      return static::isAdmin($user);
   }

   public static function getEloquentQuery(): \Illuminate\Database\Eloquent\Builder
   {
      $query = parent::getEloquentQuery();

      return static::scopeToUserIfNotAdmin($query);
   }
```

Το shouldRegisterNavigation() ορίζει ότι το resource εμφανίζεται μόνο στο διοικητικό περιβάλλον, άρα δεν υπάρχει σύνδεσμος για μη διαχειριστικές χρήσεις. Το getEloquentQuery() λειτουργεί ως δεύτερο επίπεδο ελέγχου, ώστε το query να παραμένει περιορισμένο και προβλέψιμο σε περίπτωση μελλοντικής επέκτασης ή αλλαγής κανόνων. Έτσι διασφαλίζεται ότι η πρόσβαση στα δεδομένα είναι ελεγχόμενη τόσο στο επίπεδο UI όσο και στο επίπεδο data.

5.3.4 Resource χρηστών

Η διαχείριση χρηστών αποτελεί καθαρά διοικητική λειτουργία και οργανώνεται ως ξεχωριστός πόρος στο Filament.

```
<?php
class UserResource extends Resource
{
   use ResourceAuthorization;
   protected static ?string $model = User::class;
   protected static string|BackedEnum|null $navigationIcon = Heroicon::OutlinedUsers;
   public static function getNavigationLabel(): string
   {
      return __('filament-admin.navigation.users');
   }
   public static function getNavigationGroup(): string|UnitEnum|null
   {
      return __('filament-admin.navigation_groups.administration');
```

```
    }
    public static function shouldRegisterNavigation(): bool
    {
        $user = static::currentUser();
        return static::isAdmin($user);
    }
}
```

Το UserResource συγκεντρώνει τη διαχείριση χρηστών στο admin panel, οργανωμένη στο group “Administration”. Η μέθοδος shouldRegisterNavigation() εξασφαλίζει ότι το μενού εμφανίζεται μόνο σε διαχειριστές, κρατώντας τον έλεγχο χρηστών ως αποκλειστικά διοικητική λειτουργία. Αυτό βοηθά στη διατήρηση σαφούς διαχωρισμού ανάμεσα σε admin διαδικασίες και καθημερινές ενέργειες χρηστών.

5.3.5 Resource πληρωμών.

Η διαχείριση πληρωμών οργανώνεται σε ξεχωριστή ενότητα του admin panel, ώστε τα οικονομικά δεδομένα να παραμένουν συγκεντρωμένα και ελεγχόμενα.

```
<?php
class PaymentResource extends Resource
{
    use ResourceAuthorization;
    protected static ?string $model = Payment::class;
    protected static string|BackedEnum|null $navigationIcon = Heroicon::OutlinedCreditCard;
    public static function getNavigationGroup(): string|UnitEnum|null
    {
        return __('filament-admin.navigation_groups.finance');
    }
    public static function getEloquentQuery(): \Illuminate\Database\Eloquent\Builder
    {
        $query = parent::getEloquentQuery();
        return static::scopeToUserIfNotAdmin($query);
    }
}
```

Το resource πληρωμών τοποθετείται στο group “Finance”, ώστε να είναι ξεκάθαρα διαχωρισμένο από λειτουργίες διαχείρισης χρηστών ή περιεχομένου. Το getEloquentQuery() λειτουργεί ως πρόσθετο επίπεδο ελέγχου στο query, διασφαλίζοντας ότι η πρόσβαση στα δεδομένα πληρωμών παραμένει αυστηρά καθορισμένη και έτοιμη για μελλοντική επέκταση κανόνων.

5.4 Ενοποίηση πληρωμών Stripe (cashier)

Η ενότητα αυτή δείχνει πώς η εφαρμογή συνδέεται με το Stripe μέσω του Laravel Cashier: δημιουργία συνδρομής, χειρισμός απαιτούμενης πληρωμής (3-D secure), ενημέρωση της βάσης και πρόσβαση στο billing portal. Επίσης περιλαμβάνει το client-side κομμάτι (Stripe Elements) και το webhook endpoint.

5.4.1 Δημιουργία συνδρομής (server-side, Cashier)

```
<?php
    $validated = $request->validate([
        'membership_type_id' => ['required', 'integer', 'exists:membership_types,id'],
        'payment_method' => ['required', 'string'],
    ]);

    $membershipType = MembershipType::query()->findOrFail($validated['membership_type_id']);

    if (empty($membershipType->stripe_price_id) || str_starts_with($membershipType->stripe_price_id,
'price_placeholder_')) {
        return back()->with('status', __('site.dashboard.subscriptions.missing_price_id'));
    }

    try {
        $user->newSubscription('default', $membershipType->stripe_price_id)
            ->create($validated['payment_method']);
    } catch (IncompletePayment $exception) {
        return redirect()->route('cashier.payment', [
            $exception->payment->id,
            'redirect' => route('dashboard.subscriptions'),
        ]);
    }
```

Το membership_type_id και το payment_method είναι τα δύο κρίσιμα inputs: το πρώτο καθορίζει ποιο πακέτο αγοράζεται, ενώ το δεύτερο είναι το Stripe payment method που δημιουργείται στο frontend. Ο έλεγχος stripe_price_id εξασφαλίζει ότι το πακέτο έχει αντιστοιχιστεί σωστά σε Stripe Price· αν όχι, η αγορά μπλοκάρεται για να αποφευχθούν λάθος χρεώσεις.

Η κλήση newSubscription('default', $membershipType->stripe_price_id)->create(...) μεταφέρει την ευθύνη δημιουργίας στο Stripe μέσω Cashier. Έτσι η εφαρμογή δεν χειρίζεται λεπτομέρειες χρέωσης, αλλά χρησιμοποιεί την επίσημη ροή του Stripe. Αν απαιτείται επιβεβαίωση (SCA/3-D secure), το IncompletePayment ανακατευθύνει τον χρήστη σε ειδική σελίδα όπου ολοκληρώνει την πληρωμή. Αυτό προστατεύει από "μισοτελειωμένες" συνδρομές και κρατά τον συγχρονισμό σωστό.

5.4.2 Ενημέρωση βάσης και καταγραφή πληρωμής

```
<?php
    if ($subscription && Schema::hasTable('subscriptions')) {
        $subscription->forceFill([
            'start_date' => $membershipStartDate,
            'end_date' => $membershipEndDate,
            'price' => $membershipType->price,
            'status' => 'active',
        ])->save();
    }
    if (Schema::hasTable('payments')) {
        Payment::query()->create([
            'subscription_id' => $subscription?->getKey(),
            'user_id' => $user->getKey(),
            'member_id' => optional($user->member)->getKey(),
            'amount' => $membershipType->price,
            'payment_date' => now()->toDateString(),
            'method' => 'online',
            'payment_method' => 'stripe',
            'transaction_id' => $subscription?->stripe_id,
            'status' => 'completed',
        ]);
    }
```

Μετά την επιτυχή δημιουργία στο Stripe, η εφαρμογή **ενημερώνει τοπικά** τον πίνακα subscriptions. Τα πεδία start_date, end_date, price, status κρατούν ένα επιχειρησιακό "snapshot" ώστε το dashboard να εμφανίζει κατάσταση χωρίς να κάνει συνεχή calls στο Stripe.
Στη συνέχεια καταγράφεται μια εγγραφή στον πίνακα payments. Αυτό είναι σημαντικό γιατί το Stripe δεν αντικαθιστά το εσωτερικό ιστορικό πληρωμών. Με το transaction_id (Stripe subscription id) μπορούμε να κάνουμε trace, ενώ το status επιτρέπει reporting ακόμα και αν στο μέλλον υπάρξουν αποτυχημένες ή εκκρεμείς πληρωμές. Άρα το σύστημα διατηρεί **τοπικό audit trail** ανεξάρτητο από τρίτους.

5.4.3 Διαχείριση συνδρομής μέσω cashier (cancel/resume/portal)

Οι βασικές ενέργειες συνδρομής χαρτογραφούνται απευθείας σε λειτουργίες του Cashier. Η ακύρωση και η επαναφορά γίνονται με ασφαλείς ελέγχους, ενώ το billing portal παρέχει επίσημη διαχείριση πληρωμών μέσω Stripe χωρίς επιπλέον custom UI.

Η ακύρωση γίνεται μόνο όταν υπάρχει **έγκυρη** συνδρομή. Αυτό προστατεύει από μηδενικές/ανενεργές εγγραφές. Το cancel() δεν διαγράφει τη συνδρομή, αλλά την μεταφέρει σε "grace period", επιτρέποντας πρόσβαση μέχρι το τέλος του ήδη πληρωμένου διαστήματος.
Η επαναφορά (resume()) γίνεται μόνο αν η συνδρομή βρίσκεται σε grace period. Αυτό αποτρέπει

“λάθος” επαναφορές. Το billing portal είναι η επίσημη διεπαφή του Stripe: ο χρήστης μπορεί να αλλάξει κάρτα ή στοιχεία πληρωμής χωρίς να χρειάζεται custom UI. Έτσι μειώνεται η πολυπλοκότητα και αυξάνεται η αξιοπιστία.

```
<?php
    public function destroy(Request $request): RedirectResponse
    {
        $subscription = $request->user()->subscription('default');
        if (! $subscription || ! $subscription->valid()) {
            return back()->with('status', __('site.dashboard.subscriptions.no_active_subscription'));
        }
        $subscription->cancel();
        return to_route('dashboard.subscriptions')
            ->with('status', __('site.dashboard.subscriptions.cancel_success'));
    }
    public function resume(Request $request): RedirectResponse
    {
        $subscription = $request->user()->subscription('default');
        if (! $subscription || ! $subscription->onGracePeriod()) {
            return back()->with('status', __('site.dashboard.subscriptions.not_on_grace_period'));
        }
        $subscription->resume();
        return to_route('dashboard.subscriptions')
            ->with('status', __('site.dashboard.subscriptions.resume_success'));
    }
    public function billingPortal(Request $request): RedirectResponse
    {
        return $request->user()->redirectToBillingPortal(route('dashboard.subscriptions'));
    }
```

5.4.4 Stripe Elements

Το Stripe Elements δημιουργεί ένα ασφαλές payment_method στο client, χωρίς να περνούν στοιχεία κάρτας από τον server της εφαρμογής. Το paymentMethod.id είναι το μόνο που αποστέλλεται, και αυτό είναι που χρησιμοποιεί ο Cashier για τη δημιουργία της συνδρομής.

Η διαχείριση σφαλμάτων γίνεται στο frontend (errorElement), ώστε ο χρήστης να βλέπει άμεσα τι δεν πήγε καλά. Μόνο όταν υπάρχει έγκυρο payment method γίνεται submit της φόρμας, άρα αποφεύγονται αποτυχημένα requests στο backend.

```
const stripe = Stripe(@json($stripeKey));
const elements = stripe.elements({ /* appearance config */ });
const cardElement = elements.create('card', { /* style */ });
```

```
const form = document.getElementById('subscription-form');
const paymentMethodInput = document.getElementById('payment_method');
const errorElement = document.getElementById('card-errors');

cardElement.mount('#card-element');
form.addEventListener('submit', async (event) => {
    event.preventDefault();
    errorElement.textContent = '';

    const {paymentMethod, error} = await stripe.createPaymentMethod({
        type: 'card',
        card: cardElement,
        billing_details: {
            name: @json(auth()->user()->name),
            email: @json(auth()->user()->email),
        },
    });

    if (error) {
        errorElement.textContent = error.message ?? 'Payment method creation failed.';
        return;
    }

    paymentMethodInput.value = paymentMethod.id;
    form.submit();
});
```

5.4.5 Webhook endpoints

```
<?php
Route::post('/stripe/webhook', [WebhookController::class, 'handleWebhook'])->name('cashier.webhook');
```

Το webhook endpoint επιτρέπει στο Stripe να ενημερώνει την εφαρμογή για αλλαγές στην κατάσταση (π.χ. αποτυχημένη πληρωμή, ακύρωση, επιτυχής χρέωση). Ο WebhookController του Cashier χειρίζεται αυτόματα τα περισσότερα events και ενημερώνει τις τοπικές εγγραφές. Έτσι, ακόμη κι αν μια αλλαγή συμβεί **εκτός** της εφαρμογής (π.χ. μέσα από το Stripe dashboard), το σύστημα παραμένει συγχρονισμένο και οι πληροφορίες στο dashboard παραμένουν έγκυρες.

5.5 Σχεδιασμός διεπαφής (Tailwind CSS)

Η διεπαφή υλοποιείται με Tailwind CSS σε utility-first λογική. Αυτό επιτρέπει γρήγορη σύνθεση UI, συνεπή τυπογραφία, responsive συμπεριφορά και ελεγχόμενη οπτική ταυτότητα χωρίς ξεχωριστά CSS αρχεία ανά σελίδα.

5.5.1 Ρύθμιση Tailwind

```
export default {
  content: [
    './vendor/laravel/framework/src/Illuminate/Pagination/resources/views/*.blade.php',
    './storage/framework/views/*.php',
    './resources/views/**/*.blade.php',
  ],

  theme: {
    extend: {
      fontFamily: {
        sans: ['Figtree', ...defaultTheme.fontFamily.sans],
      },
    },
  },

  plugins: [forms],
};
```

Το content ορίζει όλα τα Blade templates που πρέπει να σαρώνονται, ώστε το Tailwind να κρατά μόνο τις κλάσεις που πραγματικά χρησιμοποιούνται. Αυτό μειώνει δραστικά το τελικό CSS. Η επέκταση της fontFamily επιβάλλει συγκεκριμένη γραμματοσειρά σε όλη τη διεπαφή, ενώ το plugin @tailwindcss/forms δίνει συνεπή, “reset” εμφάνιση στις φόρμες, πάνω στην οποία χτίζεται το υπόλοιπο styling.

5.5.2 Επαναχρησιμοποιήσιμα UI blocks με @apply

```
@layer components {
  .ui-container {
    @apply mx-auto w-full max-w-7xl px-4 sm:px-6 lg:px-8;
  }

  .ui-card {
    @apply rounded-2xl border border-white/20 bg-slate-900/80 p-6 shadow-2xl shadow-black/40;
  }
```

```
  .ui-title {
    @apply text-3xl font-bold tracking-tight text-white md:text-4xl;
  }

  .ui-subtitle {
    @apply mt-2 text-sm text-slate-300 md:text-base;
  }
}
```

Με το @layer components δημιουργούνται επαναχρησιμοποιήσιμες κλάσεις που συγκεντρώνουν πολλά utilities σε "δομικά blocks" (ui-container, ui-card, ui-title). Έτσι οι σελίδες παραμένουν καθαρές, η αισθητική είναι συνεπής, και οι αλλαγές γίνονται κεντρικά. Η χρήση responsive utilities (sm, lg, md) μέσα στα @apply διατηρεί τη συμπεριφορά προσαρμοστική σε διαφορετικά μεγέθη οθόνης.

5.5.3 Παράδειγμα χρήσης utilities σε σελίδα

```
<section class="relative -mt-10 min-h-[80vh]">
  <div class="absolute inset-0">
    <img
      src="{{ asset('images/unsplash1.jpg') }}"
      alt="{{ __('site.public.home.gallery_alt_1') }}"
      class="h-full w-full object-cover"
      loading="eager"
      decoding="async"
    />
    <div class="absolute inset-0 bg-gradient-to-b from-black/40 via-slate-950/45 to-slate-950/70"></div>
  </div>

  <div class="ui-container relative z-10 flex min-h-[80vh] items-center py-16">
    <div class="max-w-2xl text-center md:text-left">
      <h1 class="text-4xl font-bold tracking-tight text-white sm:text-5xl lg:text-6xl">
        {{ __('site.public.home.title') }}
      </h1>
```

Το Hero section χρησιμοποιεί utilities για layout, βάθος και οπτική έμφαση: min-h-[80vh] για ύψος, absolute inset-0 για πλήρη κάλυψη, και gradient overlay για αναγνωσιμότητα πάνω σε εικόνα. Η τυπογραφία κλιμακώνεται με responsive κλάσεις (sm:text-5xl, lg:text-6xl), ενώ το ui-container εξασφαλίζει σταθερό πλάτος σε όλες τις σελίδες.

5.5.4 Φόρμες και συνέπεια UI

Η φόρμα αξιοποιεί το forms plugin για καθαρή βάση και στη συνέχεια εφαρμόζει utilities για σκούρο theme, rounded corners, και έντονη χρωματική έμφαση στο focus. Έτσι κάθε input έχει συνεπή εμφάνιση και "feedback" όταν ο χρήστης αλληλεπιδρά, χωρίς επιπλέον custom CSS ανά σελίδα.

```
<label class="mb-1 block text-sm font-semibold text-slate-200">
{{ __('site.public.contact.full_name') }}
</label>
<input type="text" class="w-full rounded-lg border-white/10 bg-slate-950/60 text-slate-100 focus:border-amber-400 focus:ring-amber-400/30" placeholder="{{ __('site.public.contact.name_placeholder') }}" />
```

5.6 Επίλογος κεφαλαίου

Στο κεφάλαιο αυτό παρουσιάστηκε η πρακτική υλοποίηση της πλατφόρμας, με έμφαση στις κύριες λειτουργίες του δημόσιου site, του dashboard, του admin panel και της ενοποίησης πληρωμών. Παράλληλα αναδείχθηκε ο ρόλος του Tailwind στη συνεπή σχεδίαση της διεπαφής. Με αυτό το υπόβαθρο, το επόμενο κεφάλαιο μπορεί να εστιάσει στο σύστημα προτάσεων, αξιοποιώντας τις υποδομές που ήδη έχουν περιγραφει.

Κεφάλαιο 6ο: Σύστημα προτάσεων

Το κεφάλαιο αυτό παρουσιάζει το σύστημα προτάσεων της πλατφόρμας: τους στόχους του, τους κανόνες που εφαρμόζει, τα δεδομένα που αξιοποιεί και τον τρόπο αξιολόγησης της απόδοσής του. Μέσα από επιλεγμένα αποσπάσματα κώδικα και επεξηγήσεις, αναδεικνύεται πώς το recommendation module ενσωματώνεται στο dashboard και πώς παρακολουθείται η αποτελεσματικότητά του.

6.1 Αλγόριθμοι προτάσεων

Το recommendation module υλοποιεί δύο συμπληρωματικές στρατηγικές: **rule-based** και **collaborative**. Η επιλογή έγινε ώστε να υπάρχει ισορροπία ανάμεσα στην **ερμηνευσιμότητα** (να μπορεί ο χρήστης/σύστημα να εξηγήσει γιατί προτάθηκε κάτι) και στην **αξιοποίηση συλλογικής συμπεριφοράς** (να ανακαλύπτονται προπονήσεις/μαθήματα που δεν προκύπτουν από απλούς κανόνες). Παράλληλα, υπάρχει **fallback** στρατηγική δημοφιλίας για να αντιμετωπιστεί το "cold-start" πρόβλημα (νέοι χρήστες χωρίς ιστορικό).

6.1.1 Rule-based (βασισμένα σε κανόνες) στρατηγική

Η rule-based (βασισμένα σε κανόνες) προσέγγιση στηρίζεται σε ρητούς κανόνες που παράγουν ένα σκορ ανά υποψήφιο μάθημα. Τα κύρια σήματα είναι:

- Στόχος χρήστη (primary_goal): αν το μάθημα "ταιριάζει" με τον στόχο (π.χ. cardio, strength), προστίθεται σημαντικό βάρος.
- Προτιμώμενη ώρα (preferred_time_slot): αν η ώρα του μαθήματος συμφωνεί με την προτίμηση (πρωί/απόγευμα/βράδυ), αυξάνεται το σκορ.
- Affinity με trainer: αν ο χρήστης έχει κλείσει προπόνηση/μαθήματα με συγκεκριμένο trainer στο παρελθόν, οι μελλοντικές προπονήσεις/μαθήματα του ίδιου trainer παίρνουν bonus.
- Προτίμηση ημέρας: αν ο χρήστης "συνηθίζει" να κλείνει σε συγκεκριμένη ημέρα της εβδομάδας, οι προπονήσεις/μαθήματα στην ίδια ημέρα ενισχύονται.
- Διαθεσιμότητα: προπονήσεις/μαθήματα με περισσότερες διαθέσιμες θέσεις παίρνουν μικρή ενίσχυση, ώστε να προτείνονται επιλογές που είναι πιο πιθανό να κλειστούν.

Η στρατηγική αυτή είναι **ερμηνεύσιμη**: κάθε βαθμολογία συνοδεύεται από reason_key, δηλαδή τον "κύριο λόγο" που οδήγησε στην πρόταση. Έτσι στο UI μπορεί να εμφανιστεί "πρόταση επειδή ταιριάζει στον στόχο σου" ή "επειδή κάνεις συχνά προπονήσεις/μαθήματα αυτού του trainer". Αυτό βελτιώνει την εμπιστοσύνη του χρήστη και κάνει το σύστημα πιο "διαφανές".

6.1.2 Συνεργατική διήθηση (Collaborative) στρατηγική

Η συνεργατική διήθηση (ή στα αγγλικά collaborative) στρατηγική αξιοποιεί το ιστορικό κρατήσεων όλων των χρηστών. Η λογική είναι:

- **Εντοπισμός "όμοιων" χρηστών**:
  Για κάθε χρήστη, βρίσκονται άλλοι χρήστες με **επικάλυψη** σε προπονήσεις/μαθήματα που έχουν ήδη κλείσει. Όσο μεγαλύτερη η επικάλυψη, τόσο μεγαλύτερο το βάρος τους.
- **Συσσώρευση scores**:
  Για κάθε μάθημα που έχουν κλείσει οι "όμοιοι" χρήστες, προστίθεται score ανάλογο με το βάρος τους. Αυτό δημιουργεί μια συνολική κατάταξη μαθημάτων που τείνουν να ενδιαφέρουν άτομα με παρόμοιο ιστορικό.

- **Αφαίρεση ήδη κλεισμένων προπονήσεων/μαθημάτων**:
  Οι προπονήσεις/μαθήματα που έχει ήδη κλείσει ο χρήστης αφαιρούνται από τους υποψηφίους, ώστε οι προτάσεις να είναι πραγματικά "νέες".

Η collaborative στρατηγική προσφέρει **ποικιλία** και μπορεί να φέρει στην επιφάνεια προπονήσεις/μαθήματα που ο χρήστης δεν θα ανακάλυπτε μόνο μέσω rules. Είναι ιδιαίτερα χρήσιμη όταν υπάρχουν αρκετά δεδομένα ιστορικού.

6.1.3 Εφεδρική (Fallback) στρατηγική

Αν καμία από τις δύο στρατηγικές δεν παράγει αποτέλεσμα (π.χ. νέος χρήστης χωρίς ιστορικό ή έλλειψη δεδομένων), ενεργοποιείται η εφεδρική (ή στα αγγλικά fallback) με βάση τη δημοφιλία:

- Οι προπονήσεις/μαθήματα ταξινομούνται με βάση τις επιβεβαιωμένες κρατήσεις.
- Αν ο χρήστης έχει προτιμώμενη ώρα, οι προπονήσεις/μαθήματα που ταιριάζουν στην ώρα αυτή προτάσσονται.

Έτσι το σύστημα δεν επιστρέφει ποτέ "κενό", κάτι που θα έβλαπτε την εμπειρία χρήστη.

6.1.4 Επιλογή στρατηγικής (variant)

Η επιλογή μεταξύ rule-based και collaborative δεν γίνεται τυχαία σε κάθε αίτημα. Υπολογίζεται μια **σταθερή τιμή** από το user_id (hash), ώστε ο ίδιος χρήστης να παραμένει στο ίδιο variant. Αυτό εξασφαλίζει συνέπεια στις προτάσεις και επιτρέπει αξιολόγηση των στρατηγικών σε βάθος χρόνου (quasi A/B testing).

6.1.5 Μαθηματική διατύπωση (ενδεικτική)

Για ένα χρήστη u και ένα μάθημα c, το rule-based σκορ μπορεί να περιγραφεί ως σταθμισμένο άθροισμα κριτηρίων:

Εξίσωση 6.1

Όπου:

- U=χρήστης, c=μάθημα
- N(u): σύνολο «γειτόνων» του (χρήστες με επικαλύψεις στο ιστορικό).
- , αν το μάθημα ταιριάζει στον στόχο του χρήστη, αλλιώς 0.
- ,αν η ώρα του μαθήματος ταιριάζει με το προτιμώμενο time slot.
- αριθμός πρόσφατων κρατήσεων του χρήστη στον ίδιο trainer (μέσα στους τελευταίους μήνες).
- αν το μάθημα είναι στην προτιμώμενη ημέρα.
- διαθέσιμες θέσεις (χωρητικότητα μείον επιβεβαιωμένες κρατήσεις).

Τα βάρη (35, 25, 20, 10, 10) είναι **εμπειρικά**, ώστε να δίνεται μεγαλύτερη βαρύτητα στον στόχο του χρήστη και στη χρονική προτίμηση, ενώ η διαθεσιμότητα λειτουργεί ως ήπια ενίσχυση.

Για τη collaborative στρατηγική, ορίζουμε ένα σύνολο "γειτόνων" (χρήστες με επικαλυπτόμενες κρατήσεις). Το σκορ ενός μαθήματος είναι:

Εξίσωση 6.2

όπου είναι ο αριθμός κοινών μαθημάτων στο ιστορικό κρατήσεων και αν ο u έχει κάνει κράτηση στο c, αλλιώς ισούται με το 0 . Έτσι, προπονήσεις/μαθήματα που εμφανίζονται συχνά στο ιστορικό "όμοιων" χρηστών παίρνουν υψηλότερο σκορ.

Όταν δεν υπάρχουν επαρκή δεδομένα, ενεργοποιείται fallback δημοφιλίας:

Εξίσωση 6.3

Όπου: αριθμός επιβεβαιωμένων κρατήσεων για το μάθημα, και οι προτάσεις ταξινομούνται κατά φθίνουσα δημοφιλία (με προτεραιότητα σε προπονήσεις/μαθήματα που ταιριάζουν στη χρονική προτίμηση).

Η αξιολόγηση του recommendation module βασίζεται σε μετρικές που προκύπτουν από τις καταγραφές εμφανίσεων (impressions), κλικ (clicks) και κρατήσεων (bookings). Η βασική μετρική είναι το CTR, δηλαδή:

Εξίσωση 6.4

Όπου clicks , impressions, bookings, αριθμοί κλικ, εμφανίσεων και κρατήσεων αντίστοιχα , που δείχνει πόσο ελκυστικές ήταν οι προτάσεις. Επιπλέον, η:

Εξίσωση 6.5

δείχνει πόσο συχνά μια πρόταση οδήγησε σε πραγματική ενέργεια, ενώ η :

Εξίσωση 6.6

μετρά τη μετατροπή μετά από ενδιαφέρον, άρα λειτουργεί ως δείκτης "ποιότητας" των προτάσεων.

Οι μετρικές υπολογίζονται σε συγκεκριμένο χρονικό παράθυρο (π.χ. 30 ημέρες) ώστε να αποφεύγονται παραπλανητικά συμπεράσματα από πολύ παλιά δεδομένα. Παράλληλα, καταγράφονται ανά variant (rule-based / collaborative), επιτρέποντας συγκρίσιμη αξιολόγηση των στρατηγικών. Οι τιμές χρησιμοποιούνται για βελτιστοποίηση βαρών, εντοπισμό bias και τεκμηριωμένες αποφάσεις ως προς το ποια στρατηγική αποδίδει καλύτερα σε πραγματικές κρατήσεις.

6.2 Κανόνες/Αλγόριθμοι (κριτήρια, βαρύτητες)

Στο υποκεφάλαιο αυτό αναλύονται οι κανόνες που χρησιμοποιεί το recommendation module για να δώσει βαθμολογία στις προπονήσεις/μαθήματα. Η λογική βασίζεται σε μετρήσιμα κριτήρια (στόχος, ώρα, trainer affinity, ημέρα, διαθεσιμότητα) με **συγκεκριμένα βάρη** και σε κανόνες συνεργατικής σύγκρισης (overlap χρηστών). Τα βάρη είναι εμπειρικά και μπορούν να αναπροσαρμοστούν με βάση τις μετρικές του συστήματος.

6.2.1 Rule-based κριτήρια και βάρη

Η rule-based στρατηγική δίνει αθροιστική βαθμολογία με συγκεκριμένα βάρη. Ο στόχος του χρήστη έχει το μεγαλύτερο βάρος (35), γιατί αποτελεί το πιο "σταθερό" σήμα προτίμησης. Η χρονική προτίμηση έχει σημαντικό βάρος (25), ώστε να ευνοούνται προπονήσεις/μαθήματα σε ώρες που ο χρήστης μπορεί πράγματι να παρακολουθήσει.

Η συγγένεια με trainer υπολογίζεται από το ιστορικό κρατήσεων και προσθέτει έως 20 μονάδες, με min(20, trainerScore*5) ώστε να μην "φουσκώνει" υπερβολικά το σκορ. Η προτίμηση ημέρας δίνει μικρότερο bonus (10), ενώ η διαθεσιμότητα προσθέτει έως 10 μονάδες για να προτείνονται προπονήσεις/μαθήματα που είναι πιο πιθανό να "χωράνε".

Το reason_key κρατά τον πρώτο σημαντικό λόγο που ανέβασε το σκορ (goal/time/trainer), ώστε να εμφανίζεται ερμηνεύσιμη εξήγηση στο UI.

```
<?php
$scored = $candidates->map(function (GymClass $class) use ($goal, $preferredTimeSlot, $trainerAffinity, $preferredDay):
array {
    $score = 0;
    $reasons = [];
    if ($this->classMatchesGoal($class, $goal)) {
        $score += 35;
        $reasons[] = 'goal';
    }
    if ($this->classMatchesTimeSlot($class, $preferredTimeSlot)) {
        $score += 25;
        $reasons[] = 'time';
    }
    $trainerScore = (int) ($trainerAffinity[$class->trainer_id] ?? 0);
    if ($trainerScore > 0) {
        $score += min(20, $trainerScore * 5);
        $reasons[] = 'trainer';
    }
    if ($preferredDay && $class->start_time?->dayOfWeekIso === $preferredDay) {
        $score += 10;
    }
    $availabilityBoost = max(0, ($class->capacity - (int) $class->confirmed_bookings_count));
    $score += min(10, $availabilityBoost);
    return [
        'gym_class' => $class,
        'score' => $score,
        'reason_key' => $reasons[0] ?? 'popular',
    ];
});
```

6.2.2 Κριτήρια από ιστορικό (trainer affinity & preferred day)

Το σύστημα δεν χρησιμοποιεί όλο το ιστορικό, αλλά **παράθυρο 6 μηνών**, ώστε οι προτιμήσεις να αντανακλούν πρόσφατη συμπεριφορά.
Το trainerAffinity υπολογίζεται ως πλήθος κρατήσεων ανά trainer, άρα λειτουργεί σαν "ήπια προτίμηση" προς συγκεκριμένους trainers. Παράλληλα, το preferredDay εξάγεται από τις ημέρες κρατήσεων που εμφανίζονται συχνότερα. Με αυτόν τον τρόπο οι κανόνες δεν βασίζονται μόνο σε δηλωμένες προτιμήσεις, αλλά και σε πραγματική συμπεριφορά.

```
<?php
```

```
$recentConfirmedBookings = Booking::query()
    ->with('gymClass:id,trainer_id')
    ->where('user_id', $user->getKey())
    ->where('status', 'confirmed')
    ->where('booking_date', '>=', now()->subMonths(6))
    ->get();

$trainerAffinity = $recentConfirmedBookings
    ->pluck('gymClass.trainer_id')
    ->filter()
    ->countBy();

$preferredDay = $recentConfirmedBookings
    ->pluck('booking_date')
    ->filter()
    ->map(fn ($date) => $date->dayOfWeekIso)
    ->countBy()
    ->sortDesc()
    ->keys()
    ->first();
```

6.2.3 Collaborative κανόνας (overlap χρηστών)

```
<?php
$peerRows = Booking::query()
    ->selectRaw('user_id, COUNT(DISTINCT class_id) as overlap')
    ->where('status', 'confirmed')
    ->whereIn('class_id', $targetClassIds, 'and', false)
    ->where('user_id', '!=', $user->getKey())
    ->groupBy('user_id')
    ->orderByDesc('overlap')
    ->limit(20)
    ->get();
$peerWeights = $peerRows
    ->mapWithKeys(fn ($row) => [(int) $row->user_id => (int) $row->overlap]);
$peerClassScores = Booking::query()
    ->where('status', 'confirmed')
    ->whereIn('user_id', $peerWeights->keys(), 'and', false)
    ->whereIn('class_id', $bookedClassIds, 'and', true)
```

```
    ->get(['user_id', 'class_id'])
    ->reduce(function (array $scores, Booking $booking) use ($peerWeights): array {
        $classId = (int) $booking->class_id;
        $userWeight = (int) ($peerWeights[(int) $booking->user_id] ?? 0);
        if (! isset($scores[$classId])) {
            $scores[$classId] = 0;
        }
        $scores[$classId] += $userWeight;
        return $scores;
    }, []);
```

Η collaborative στρατηγική αναζητά χρήστες με **επικάλυψη** στις κρατήσεις (overlap). Όσο μεγαλύτερη η επικάλυψη, τόσο μεγαλύτερο το βάρος που δίνεται στον "γείτονα". Οι προπονήσεις/μαθήματα που έχουν κλείσει αυτοί οι γείτονες προσθέτουν score, ανάλογο με το βάρος τους. Αυτό σημαίνει ότι ένα μάθημα το οποίο εμφανίζεται συχνά σε χρήστες με υψηλή επικάλυψη συγκεντρώνει υψηλότερο score.
Ο κανόνας αυτός επιτρέπει στο σύστημα να **ανακαλύπτει** προπονήσεις/μαθήματα που ταιριάζουν στη "συλλογική" συμπεριφορά, ακόμα κι αν δεν έχουν ευθεία σύνδεση με τις δηλωμένες προτιμήσεις του χρήστη.

6.2.4 Fallback δημοφιλιάς

```
<?php
$classes = $this->upcomingAvailableClasses($bookedClassIds)
    ->sortByDesc('confirmed_bookings_count');
if ($preferredTimeSlot) {
    $matching = $classes->filter(fn (GymClass $class) => $this->classMatchesTimeSlot($class, $preferredTimeSlot));
    if ($matching->isNotEmpty()) {
        $classes = $matching->concat($classes->diff($matching));
    }
}
return $classes
    ->take($limit)
    ->map(fn (GymClass $class): array => [
        'gym_class' => $class,
        'score' => (int) $class->confirmed_bookings_count,
        'reason_key' => 'popular',
    ])
    ->values();
```

Το fallback ενεργοποιείται όταν οι άλλες στρατηγικές δεν δίνουν αποτέλεσμα. Οι προπονήσεις/μαθήματα ταξινομούνται με βάση τις επιβεβαιωμένες κρατήσεις, άρα επιλέγονται τα πιο δημοφιλή. Αν υπάρχει προτιμώμενη ώρα, αυτά τα προπονήσεις/μαθήματα μεταφέρονται στην κορυφή της λίστας χωρίς να αλλοιώνεται η συνολική κατάταξη. Έτσι διατηρείται **συνέπεια** και αποφεύγεται "κενό" UI ακόμη και σε νέους χρήστες.

6.3 Δεδομένα εισόδου και παρακολούθηση (Impressions/clicks/attribution)

Το recommendation module χρειάζεται δύο κατηγορίες δεδομένων: (α) δεδομένα προφίλ/προτιμήσεων του χρήστη και (β) δεδομένα συμπεριφοράς (κρατήσεις και αλληλεπίδραση με προτάσεις). Στην ενότητα αυτή φαίνεται πώς συλλέγονται και πώς καταγράφονται οι ενέργειες ώστε να είναι μετρήσιμη η απόδοση των προτάσεων.

Τα δεδομένα που προστέθηκαν είναι δοκιμαστικά και χρησιμοποιήθηκαν αποκλειστικά για να επαληθευτεί η λειτουργία του site και του recommendation module. Επειδή δεν υπήρχαν πραγματικά δεδομένα χρήσης, δημιουργήθηκαν ρεαλιστικά σενάρια (προπονήσεις, trainers, προτιμήσεις και καταγραφές impressions/clicks/bookings) ώστε να μπορεί να ελεγχθεί ότι οι ροές και οι μετρικές υπολογίζονται σωστά. Συνεπώς οι τιμές αποτελούν ένδειξη λειτουργίας και όχι αντιπροσωπευτικά στατιστικά παραγωγής.

6.3.1 Εισαγωγή προτιμήσεων χρήστη

```
<?php
$validated = $request->validate([
   'age' => ['nullable', 'integer', 'min:1', 'max:150'],
   'sex' => ['nullable', Rule::in(['male', 'female', 'other'])],
   'primary_goal' => ['nullable', Rule::in(['strength', 'cardio', 'flexibility', 'balance', 'endurance', 'weight_loss', 'general_fitness'])],
   'fitness_level' => ['nullable', Rule::in(['beginner', 'intermediate', 'advanced'])],
   'preferred_time_slot' => ['nullable', Rule::in(['morning', 'afternoon', 'evening'])],
]);
if (Schema::hasTable('members')) {
   $primaryGoal = $validated['primary_goal'] ?? null;
   $fitnessLevel = $validated['fitness_level'] ?? null;
   $preferredTimeSlot = $validated['preferred_time_slot'] ?? null;

   $hasPreferenceInput = filled($primaryGoal) || filled($fitnessLevel) || filled($preferredTimeSlot);
   $member = Member::query()->where('user_id', $user->getKey())->first();

   if ($member || $hasPreferenceInput) {
      $member ??= new Member(['user_id' => $user->getKey()]);
      $member->primary_goal = $primaryGoal ?: null;
      $member->fitness_level = $fitnessLevel ?: null;
      $member->preferred_time_slot = $preferredTimeSlot ?: null;
```

```
        $member->save();
    }
}
```

Οι προτιμήσεις του χρήστη (στόχος, επίπεδο, προτιμώμενη ώρα) αποθηκεύονται στο members profile. Η χρήση validation rules περιορίζει τις τιμές σε συγκεκριμένα enums, ώστε το recommendation module να δουλεύει με καθαρό και προβλέψιμο input. Επιπλέον, αποφεύγεται η δημιουργία "άδειου" member όταν δεν υπάρχουν προτιμήσεις. Αυτό κρατά τα δεδομένα συνεπή και μειώνει noise στο scoring.

6.3.2 Καταγραφή εμφανίσεων (impressions)

```
<?php
$recommendations = $this->recommendationService->getForUser($user, limit: 3, source: 'dashboard');
if (Schema::hasTable('recommendation_impressions') && $recommendations->isNotEmpty()) {
    $recommendations = $recommendations
        ->values()
        ->map(function (array $recommendation, int $index) use ($user): array {
            $classId = (int) ($recommendation['gymClass']?->id ?? 0);
            if ($classId <= 0) {
                $recommendation['impression_id'] = null;
                return $recommendation;
            }
            $impression = RecommendationImpression::query()->create([
                'user_id' => $user->getKey(),
                'class_id' => $classId,
                'variant' => $recommendation['variant'] ?? 'rule_based',
                'source' => $recommendation['source'] ?? 'dashboard',
                'rank' => $index + 1,
                'reason_key' => $recommendation['reason_key'] ?? null,
                'shown_at' => now(),
            ]);
            $recommendation['impression_id'] = $impression->id;
            return $recommendation;
        }); }
```

Κάθε φορά που εμφανίζονται προτάσεις, δημιουργείται εγγραφή impression. Αυτή περιέχει τον χρήστη, το μάθημα, το variant (rule-based ή συνεργατική διήθηση), την πηγή (dashboard), τη σειρά εμφάνισης (rank) και τον βασικό λόγο (reason_key). Έτσι υπάρχει πλήρης "στιγμιότυπο" της πρότασης τη στιγμή που εμφανίστηκε, κάτι κρίσιμο για μελλοντική αξιολόγηση (π.χ. CTR ανά variant ή ανά θέση).

6.3.3 Καταγραφή clicks

```
<?php
public function trackRecommendationClick(Request $request): JsonResponse
{
    if (! Schema::hasTable('recommendation_impressions') || ! Schema::hasTable('recommendation_clicks')) {
        return response()->json(['ok' => false, 'message' => 'Tracking unavailable.'], 503);
    }
    $validated = $request->validate([
        'impression_id' => ['required', 'integer'],
    ]);
    $impression = RecommendationImpression::query()
        ->whereKey($validated['impression_id'])
        ->where('user_id', $request->user()->getKey())
        ->first();

    if (! $impression) {
        return response()->json(['ok' => false, 'message' => 'Impression not found.'], 404);
    }

    $click = RecommendationClick::query()->create([
        'impression_id' => $impression->id,
        'user_id' => $request->user()->getKey(),
        'clicked_at' => now(),
    ]);

    return response()->json([
        'ok' => true,
        'click_id' => $click->id,
    ]);
}
```

Τα clicks καταγράφονται μόνο όταν υπάρχει έγκυρο impression για τον συγκεκριμένο χρήστη. Αυτό προστατεύει από "ψεύτικες" εγγραφές και διασφαλίζει ότι κάθε click συνδέεται με πραγματική εμφάνιση. Η επιστροφή click_id επιτρέπει περαιτέρω χρήση/ανάλυση και αποτελεί βάση για CTR και conversion rate.

6.3.4 Attribution: Σύνδεση κράτησης με πρόταση

Το attribution συνδέει την τελική κράτηση με το impression και (αν υπάρχει) το click. Έτσι η εφαρμογή μπορεί να απαντήσει στο ερώτημα: "Αυτή η κράτηση προήλθε από ποια πρόταση;". Το updateOrCreate εξασφαλίζει ότι κάθε booking έχει **ένα μοναδικό attribution**, αποφεύγοντας διπλομετρήσεις. Παράλληλα αποθηκεύονται και τα variant και source, ώστε να μπορούν να εξαχθούν μετρικές ανά στρατηγική και ανά κανάλι.

```
<?php
private function attachBookingAttribution(Booking $booking, ?int $impressionId): void
{   if (! $impressionId || $impressionId <= 0) {
        return;  }
    if (! Schema::hasTable('booking_attributions') || ! Schema::hasTable('recommendation_impressions')) {
        return;    }
    $impression = RecommendationImpression::query()
        ->whereKey($impressionId)
        ->where('user_id', $booking->user_id)
        ->where('class_id', $booking->class_id)
        ->first();
    if (! $impression) {
        return;
    }
    $click = RecommendationClick::query()
        ->where('impression_id', $impression->id)
        ->where('user_id', $booking->user_id)
        ->latest('clicked_at')
        ->first();
    BookingAttribution::query()->updateOrCreate(
        ['booking_id' => $booking->id],
        [
            'impression_id' => $impression->id,
            'click_id' => $click?->id,
            'user_id' => $booking->user_id,
            'class_id' => $booking->class_id,
            'variant' => $impression->variant,
            'source' => $impression->source,
            'booked_at' => now(),
        ]   );}
```

6.4 Μετρικές αξιολόγησης και αποτελέσματα

Το recommendation module αξιολογείται με μετρικές που βασίζονται σε πραγματική συμπεριφορά χρήστη. Οι καταγραφές impressions, clicks και bookings επιτρέπουν να μετρηθεί τόσο η "ελκυστικότητα" των προτάσεων όσο και η τελική μετατροπή σε κράτηση. Στο παρακάτω απόσπασμα φαίνεται η συγκέντρωση των μετρικών ανά χρήστη και ανά variant.

6.4.1 Υπολογισμός μετρικών (personal + variant)

Οι μετρικές υπολογίζονται σε χρονικό παράθυρο (π.χ. 30 ημέρες), ώστε να αντικατοπτρίζουν πρόσφατη συμπεριφορά και να αποφεύγονται στρεβλώσεις από παλιά δεδομένα. Η λογική ενεργοποιείται μόνο όταν υπάρχει πλήρες tracking (impressions/clicks/attributions). Παράγονται δύο ομάδες μετρικών:

- **Personal summary**: μετρικές ανά χρήστη (impressions, clicks, bookings, CTR και conversion rates).
- **Variants summary**: συγκεντρωτικά αποτελέσματα ανά στρατηγική (rule-based / συνεργατική διήθηση).

Έτσι γίνεται δυνατή τόσο η προσωπική αξιολόγηση όσο και η σύγκριση στρατηγικών σε επίπεδο συστήματος.

```
<?php
$metricsWindowDays = 30;
$metricsFrom = now()->subDays($metricsWindowDays);
$metricsTo = now();
$isAdmin = method_exists($user, 'isAdmin') && $user->isAdmin();
$hasRecommendationTracking = Schema::hasTable('recommendation_impressions')
   && Schema::hasTable('recommendation_clicks')
   && Schema::hasTable('booking_attributions');
$recommendationMetrics = null;
if ($isAdmin) {
   $recommendationMetrics = [
      'windowDays' => $metricsWindowDays,
      'personal' => $hasRecommendationTracking
         ? $this->recommendationMetricsService->personalSummary($user, $metricsFrom, $metricsTo, 'dashboard')
         : [
            'impressions' => 0,
            'clicks' => 0,
            'bookings' => 0,
            'ctr' => 0.0,
            'impression_to_booking_rate' => 0.0,
            'click_to_booking_rate' => 0.0,
         ],
      'variants' => $hasRecommendationTracking
```

```
        ? $this->recommendationMetricsService->summaryByVariant($metricsFrom, $metricsTo, 'dashboard')
        : collect(),
];}
```

6.4.2 Χρήση μετρικών και ενδεικτικά αποτελέσματα

Στην υλοποίηση χρησιμοποιήθηκαν οι μετρικές impressions, clicks, bookings, καθώς και οι παράγωγες CTR, impression_to_booking_rate και click_to_booking_rate. Στο τρέχον δοκιμαστικό dataset καταγράφηκαν 533 impressions, 187 clicks και 66 bookings, που αντιστοιχούν σε CTR 35.08%, impression-to-booking 12.38% και click-to-booking 35.29%. Οι τιμές αυτές είναι ενδεικτικές και χρησιμοποιούνται για να επιβεβαιωθεί η σωστή λειτουργία του συστήματος, καθώς τα δεδομένα είναι συνθετικά και όχι παραγωγικά.

Οι μετρικές δεν είναι απλώς αναφορές, αλλά εργαλείο βελτίωσης. Τα όρια για "χαμηλό/υψηλό" δεν είναι απόλυτα και εξαρτώνται από το περιβάλλον του συστήματος· ωστόσο ενδεικτικά μπορούμε να θεωρήσουμε CTR χαμηλό όταν είναι <5%, μέτριο 5–15% και υψηλό >15–20%. Για conversion (impression-to-booking) ενδεικτικά χαμηλό είναι <1–2%, μέτριο 2–5% και υψηλό >5–10%, ενώ για click-to-booking χαμηλό <10%, μέτριο 10–30% και υψηλό >30%.

- Αν το CTR είναι χαμηλό, χρειάζεται βελτίωση στη σχετικότητα ή στην παρουσίαση.
- Αν το CTR είναι υψηλό αλλά το conversion χαμηλό, σημαίνει ότι οι προτάσεις είναι ενδιαφέρουσες αλλά όχι πρακτικές (π.χ. χρονικά μη βολικές ή γεμάτες).
- Η σύγκριση ανά variant επιτρέπει να αξιολογηθεί αν η rule-based ή η συνεργατική διήθηση αποδίδει καλύτερα. Με αυτή τη διαδικασία, το recommendation module μπορεί να εξελίσσεται βάσει πραγματικών δεδομένων και όχι μόνο εμπειρικών ρυθμίσεων.

6.5 Ενσωμάτωση στο site (UI/ροές χρήστη)

Το recommendation module ενσωματώνεται στο dashboard με τρόπο που συνδέει άμεσα την παρουσίαση προτάσεων με ενέργεια (κράτηση) και με παρακολούθηση (click/impression/attribution). Η ροή είναι: προβολή προτάσεων → κλικ → κράτηση → απόδοση (attribution).

6.5.1 Προβολή προτάσεων στο dashboard

Η πρόταση δεν εμφανίζεται ως απλό μάθημα αλλά συνοδεύεται από "αιτιολόγηση" (reason_key), ώστε ο χρήστης να καταλαβαίνει γιατί το σύστημα το προτείνει. Αυτό ενισχύει την ερμηνευσιμότητα του αλγορίθμου και βελτιώνει την εμπιστοσύνη στο UI. Παράλληλα, η εμφάνιση της διάρκειας και του trainer προσφέρει άμεσο context πριν την κράτηση.

```
@foreach ($recommendations as $recommendation)
  @php
    $class = $recommendation['gymClass'];
    $duration = ($class->start_time && $class->end_time)
      ? __('site.public.classes.duration_minutes', ['minutes' => $class->start_time->diffInMinutes($class->end_time)])
      : __('site.public.classes.duration_tbd');
    $reasonLabel = __('site.dashboard.index.reasons.' . $recommendation['reason_key']);
  @endphp
```

```
<x-ui.card class="space-y-3">
    <p class="text-lg font-semibold text-white">{{ $class->name }}</p>
    <p class="rounded-lg border border-amber-400/30 bg-amber-500/10 px-3 py-2 text-xs font-semibold text-amber-200">
        {{ __('site.dashboard.index.recommendation_reason') }}: {{ $reasonLabel }}
    </p>
```

6.5.2 Κράτηση από το UI (CTA → backend)

```
<form method="POST" action="{{ route('dashboard.bookings.store') }}">
    @csrf
    <input type="hidden" name="class_id" value="{{ $class->id }}">
    @if (! empty($recommendation['impression_id']))
        <input type="hidden" name="recommendation_impression_id" value="{{ $recommendation['impression_id'] }}">
    @endif
    <x-ui.button type="submit" class="w-full justify-center" data-rec-click="{{ $recommendation['impression_id'] ?? '' }}">
        {{ __('site.dashboard.index.recommendation_cta') }}
    </x-ui.button>
</form>
```

Το CTA δεν οδηγεί σε ξεχωριστή σελίδα, αλλά δημιουργεί κράτηση απευθείας μέσω POST. Το class_id είναι το βασικό input, ενώ το recommendation_impression_id συνδέει την κράτηση με τη συγκεκριμένη πρόταση. Έτσι το UI μεταφέρει στο backend όλη την πληροφορία που χρειάζεται για να γίνει attribution αργότερα.

6.5.3 Ροή κράτησης

Η ενέργεια του κουμπιού χαρτογραφείται σε συγκεκριμένη route του dashboard. Αυτό διατηρεί τη ροή εντός προστατευμένου περιβάλλοντος και επιτρέπει στον controller να εφαρμόσει όλους τους απαραίτητους ελέγχους.

```
<?php
Route::post('/bookings', [DashboardPageController::class, 'storeBooking'])->name('bookings.store');
```

6.5.4 Καταγραφή clicks (frontend tracking)

```
const clickUrl = @json(route('dashboard.recommendations.click'));
const csrfToken = @json(csrf_token());
const links = document.querySelectorAll('[data-rec-click]');

links.forEach((link) => {
    link.addEventListener('click', () => {
        const impressionId = link.getAttribute('data-rec-click');
        if (! impressionId) return;
```

```
    const payload = new FormData();
    payload.append('_token', csrfToken);
    payload.append('impression_id', impressionId);

    if (navigator.sendBeacon) {
      navigator.sendBeacon(clickUrl, payload);
      return;
    }

    fetch(clickUrl, {
      method: 'POST',
      headers: {
        'X-CSRF-TOKEN': csrfToken,
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ impression_id: impressionId }),
      keepalive: true,
    });
  });
});
```

Το click tracking αποστέλλει impression_id στον server ώστε να καταγράφεται η αλληλεπίδραση. Η χρήση sendBeacon επιτρέπει την αποστολή ακόμη και αν ο χρήστης οδηγηθεί άμεσα σε άλλη σελίδα (π.χ. λόγω submit). Αν δεν υπάρχει υποστήριξη, γίνεται fallback σε fetch. Έτσι εξασφαλίζεται αξιόπιστη καταγραφή clicks χωρίς να "σπάει" η εμπειρία χρήστη.

6.5.5 Απόδοση κράτησης σε πρόταση (backend)

Μετά τη δημιουργία της κράτησης, το σύστημα προσπαθεί να συνδέσει το booking με την αντίστοιχη πρόταση. Αυτό είναι το σημείο που ολοκληρώνεται η ροή: **πρόταση → click → κράτηση → attribution**. Χωρίς αυτό το βήμα δεν θα μπορούσαν να υπολογιστούν με ακρίβεια τα conversion metrics.

```
<?php
$booking = Booking::query()->create([
  'user_id' => $user->getKey(),
  'class_id' => $class->getKey(),
  'booking_date' => $class->start_time,
  'status' => 'confirmed',
]);
```

```
$this->attachBookingAttribution(
    $booking,
    isset($validated['recommendation_impression_id'])
        ? (int) $validated['recommendation_impression_id']
        : null,);
```

6.6 Επίλογος κεφαλαίου

Το κεφάλαιο αυτό παρουσίασε σε βάθος τον τρόπο λειτουργίας του recommendation module: τους αλγορίθμους, τα κριτήρια, τα δεδομένα εισόδου, τον μηχανισμό παρακολούθησης και τις μετρικές αξιολόγησης. Παράλληλα, φάνηκε πώς η λειτουργία ενσωματώνεται στο UI και οδηγεί σε μετρήσιμες ενέργειες (clicks/κρατήσεις). Με αυτά τα στοιχεία, το σύστημα προτάσεων τεκμηριώνεται τόσο θεωρητικά όσο και πρακτικά

Κεφάλαιο 7ο: Συμπεράσματα και Προτάσεις Βελτίωσης

7.1 Συμπεράσματα

Η εργασία ολοκλήρωσε τον στόχο της ανάπτυξης μιας ολοκληρωμένης πλατφόρμας διαχείρισης γυμναστηρίου με σύγχρονη τεχνολογική υποδομή. Υλοποιήθηκε πλήρης ροή για δημόσιο site, dashboard χρήστη, admin panel και συνδρομές/πληρωμές μέσω Stripe, ενώ η αρχιτεκτονική διατηρήθηκε καθαρή και επεκτάσιμη με βάση το MVC και τις πολιτικές πρόσβασης.

Ιδιαίτερη προστιθέμενη αξία προσφέρει το recommendation module, το οποίο συνδυάζει rule-based και συνεργατική διήθηση, καταγράφει impressions/clicks/bookings και επιτρέπει αξιολόγηση με μετρικές (CTR, conversion). Με αυτό τον τρόπο το σύστημα δεν περιορίζεται σε στατική παρουσίαση, αλλά εισάγει μηχανισμό εξατομίκευσης και βελτιστοποίησης.

Παράλληλα, η χρήση Tailwind και component-based UI εξασφάλισε συνεπή διεπαφή, ενώ το Laravel Cashier επέτρεψε ασφαλή και τυποποιημένη ενοποίηση πληρωμών. Συνολικά, το αποτέλεσμα είναι μια λειτουργική και επεκτάσιμη πλατφόρμα που καλύπτει τις βασικές ανάγκες διαχείρισης και μπορεί να εξελιχθεί με βάση πραγματικά δεδομένα χρήσης.

7.2 Περιορισμοί

Παρότι η πλατφόρμα υλοποιεί πλήρη ροή λειτουργιών, υπάρχουν ορισμένοι περιορισμοί που πρέπει να αναφερθούν. Πρώτον, η αξιολόγηση του recommendation module βασίστηκε σε συνθετικά/δοκιμαστικά δεδομένα και όχι σε πραγματική παραγωγική χρήση, άρα οι μετρικές αποτελούν ένδειξη λειτουργίας και όχι γενικεύσιμα στατιστικά.

Δεύτερον, η απουσία μεγάλης κλίμακας χρηστών και πραγματικού φορτίου σημαίνει ότι δεν μπορεί να εκτιμηθεί με ακρίβεια η απόδοση του συστήματος σε επίπεδο κλιμάκωσης (π.χ. μεγάλος αριθμός ταυτόχρονων κρατήσεων).

Τρίτον, η ενοποίηση Stripe έγινε σε περιβάλλον δοκιμών (test mode), γεγονός που δεν επιτρέπει πλήρη έλεγχο πραγματικών οικονομικών ροών και συμπεριφοράς σε περιπτώσεις αποτυχίας πληρωμής.

Τέλος, το recommendation module βασίζεται σε rule-based και συνεργατική διήθηση χωρίς χρήση προχωρημένων μοντέλων ML· αυτό καθιστά το σύστημα ερμηνεύσιμο και απλό, αλλά πιθανόν λιγότερο ισχυρό σε πιο σύνθετα σενάρια.

7.3 Μελλοντικές βελτιώσεις

Με βάση τα αποτελέσματα της υλοποίησης, προκύπτουν σαφείς δυνατότητες εξέλιξης. Πρώτον, η συλλογή πραγματικών δεδομένων χρήσης θα επιτρέψει αξιόπιστη αξιολόγηση των μετρικών και δυναμική προσαρμογή των βαρών του rule-based μοντέλου. Δεύτερον, μπορεί να ενσωματωθεί πιο προχωρημένη μορφή collaborative filtering ή μοντέλα μηχανικής μάθησης (π.χ. matrix factorization), ώστε οι προτάσεις να αξιοποιούν μεγαλύτερο εύρος συμπεριφορικών δεδομένων.

Επιπλέον, μπορεί να προστεθεί A/B testing με πραγματικούς χρήστες, ώστε να συγκρίνεται πειραματικά η απόδοση διαφορετικών στρατηγικών. Στο κομμάτι του UI, θα μπορούσαν να ενσωματωθούν πιο πλούσιες μορφές εξήγησης (“explainability”) για κάθε πρόταση, ενισχύοντας τη διαφάνεια.

Τέλος, σε επίπεδο λειτουργίας, η πλήρης μετάβαση σε production περιβάλλον θα δώσει τη δυνατότητα ελέγχου απόδοσης σε υψηλό φορτίο, πραγματικές πληρωμές και συνεχή παρακολούθηση χρηστών, ολοκληρώνοντας την ωρίμανση της πλατφόρμας.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] “Laravel - The clean stack for Artisans and agents,” Laravel. [Online]. Available: https://laravel.com/

[2] “Build Laravel apps & admin panels fast - Filament.”. [Online]. Available: https://filamentphp.com/

[3] “Starter Kits | Laravel 10.x - The clean stack for Artisans and agents,” Laravel [Online]. Available: https://laravel.com/docs/13.x/starter-kits

[4] “Laravel Herd,” Laravel Herd. [Online]. Available: https://herd.laravel.com/windows

[5]“Laravel Cashier (Stripe) | Laravel 13.x - The clean stack for Artisans and agents,” Laravel. [Online]. Available: https://laravel.com/docs/13.x/billing

[6]“Vite,” vitejs. Accessed: [Online]. Available: https://vite.dev

[7] “Visual Studio Code - The open source AI code editor | Your home for multi-agent development.” Accessed: [Online]. Available: https://code.visualstudio.com/

[8] Unsplash, “Photo by Risen Wang on Unsplash.” [Online]. Available: https://unsplash.com/photos/gym-equipment-inside-room-20jX9b35r_M

[9] Y. Koren, R. Bell, and C. Volinsky, “Matrix Factorization Techniques for Recommender Systems,” *Computer*, vol. 42, no. 8, pp. 30–37, Aug. 2009, doi: 10.1109/MC.2009.263.

[10] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, “Collaborative filtering recommender systems,” in The Adaptive Web, P. Brusilovsky, A. Kobsa, and W. Nejdl, Eds. Berlin, Germany: Springer, 2007, pp. 291–324.