Πανεπιστήμιο Πελοποννήσου
Τμήμα Πληροφορικής και Τηλεπικοινωνίων

Σχεδίαση και Ανάπτυξη Εφαρμογής Αυτοαξιολόγησης Φοιτητών

Χριστοδουλίδης Νικόλαος
Α.Μ 2011031

Επιβλέπων: Νίκος Τσελίκας – Καθηγητής

Διπλωματική Εργασία στο Π.Μ.Σ. στην Επιστήμη Υπολογιστών

Δεκέμβριος 2025

University of Peloponnese
Department of Information and Telecommunications

Design and Development of a Mobile Application to Support Students' Self-Assessment

Christodoulidis Nick
A.M 2011031

Supervisor: Nikolaos Tselikas – Professor

Diploma thesis for M.Sc. program in "Computer Science"

December 2025

Περίληψη

Τίτλος: Σχεδίαση και Ανάπτυξη Εφαρμογής Κινητού Αυτοαξιολόγησης Μαθητών

Η παρούσα διπλωματική εργασία παρουσιάζει τον σχεδιασμό και την ανάπτυξη μιας εφαρμογής για κινητές συσκευές, η οποία υποστηρίζει την αυτοαξιολόγηση των φοιτητών μέσω διαδραστικών τεστ γνώσεων. Η εφαρμογή, με την ονομασία Edutest, αναπτύχθηκε χρησιμοποιώντας τη γλώσσα προγραμματισμού Kotlin και το περιβάλλον Jetpack Compose για το λειτουργικό σύστημα Android, ενώ η Firebase Realtime Database χρησιμοποιείται ως υποδομή αποθήκευσης και διαχείρισης δεδομένων στο νέφος.

Το σύστημα επιτρέπει στους χρήστες να ελέγχουν τις γνώσεις τους σε διάφορα γνωστικά αντικείμενα και υποκατηγορίες, ενώ ο διαχειριστής έχει τη δυνατότητα να προσθέτει, να τροποποιεί ή να διαγράφει μαθήματα, ερωτήσεις και επιλογές απαντήσεων απευθείας από το περιβάλλον της εφαρμογής. Με αυτόν τον τρόπο, η ενημέρωση του περιεχομένου γίνεται σε πραγματικό χρόνο, χωρίς να απαιτείται αναβάθμιση ή επανεγκατάσταση της εφαρμογής από τους χρήστες.

Επιπλέον, ενσωματώθηκε σύστημα ελέγχου ταυτότητας χρηστών, ώστε η πρόσβαση στις λειτουργίες διαχείρισης να περιορίζεται αποκλειστικά στους διαχειριστές, ενώ η λειτουργία του πίνακα κατάταξης (leaderboard) ενισχύει τα κίνητρα των φοιτητών μέσω της παρουσίασης των κορυφαίων επιδόσεων.

Η εργασία καταδεικνύει την αποτελεσματικότητα των εφαρμογών κινητών συσκευών που βασίζονται σε τεχνολογίες νέφους για την ενίσχυση της αυτοκατευθυνόμενης μάθησης και την υποστήριξη της εκπαιδευτικής διαδικασίας μέσω ευέλικτων και δυναμικών εργαλείων αξιολόγησης.

Λέξεις-Κλειδιά (Keywords):

Εφαρμογή για κινητά, αυτοαξιολόγηση, ηλεκτρονική μάθηση, Firebase, Android, Kotlin, Jetpack Compose, σύστημα κουίζ, αξιολόγηση φοιτητών, εκπαιδευτική τεχνολογία

Abstract

Title: Design and Development of a Mobile Application to Support Students' Self-Assessment

This thesis presents the design and development of a mobile application that supports the self-assessment of students through interactive quizzes. The application was developed using **Kotlin** and **Jetpack Compose** for Android, with **Firebase Realtime Database** serving as the backend for dynamic data management.

The system enables users to test their knowledge across multiple subjects and subcategories, while administrators can add, update, or delete subjects, questions, and options directly from within the app. This approach allows for real-time content updates without requiring users to reinstall or update the application. Additionally, the system incorporates user authentication to restrict administrative access, ensuring secure data handling. A leaderboard feature further enhances student motivation by displaying the top scores and tracking user performance.

The project demonstrates the effectiveness of cloud-integrated mobile applications in promoting self-directed learning, providing both students and educators with a flexible and scalable assessment tool.

Keywords:

Mobile application, self-assessment, e-learning, Firebase, Android, Kotlin, Jetpack Compose, quiz system, student evaluation, educational technology

Contents

Κεφάλαιο 1: Εισαγωγή

1.1 Σκοπός και στόχοι της εργασίας

Η παρούσα εργασία έχει σαν στόχο τη δημιουργία μίας εκπαιδευτικής εφαρμογής αυτοαξιολόγησης. Αυτό επιτυγχάνεται με τη συμμετοχή των χρηστών σε τεστ ερωτήσεων πολλαπλής επιλογής για κάθε γνωστικό αντικείμενο που θα θεσπίσουν οι διαχειριστές του συστήματος.

Συγκεκριμένα, γίνεται αξιοποίηση της κινητής τεχνολογίας ώστε για παράδειγμα οι φοιτητές ή οποιοσδήποτε χρήστης της εφαρμογής να έχει την ελευθέρια στον προσωπικό του χρόνο και οπουδήποτε βρίσκεται να αυτοαξιολογείται σε γνωστικά αντικείμενα που έχει ήδη μελετήσει ή παρακολουθήσει, ώστε να εντοπίσει τις αδυναμίες του και τις δυνατότητές του και να οργανώνει την περαιτέρω πορεία της μελέτης του ανάλογα, προάγοντας έτσι την ενεργητική μάθηση.

Ειδικότερα, επιδίωξη της εφαρμογής είναι:

- Να παρέχει ένα δυναμικό περιβάλλον αξιολόγησης, στο οποίο οι ερωτήσεις και τα γνωστικά αντικείμενα να μπορούν να ανανεώνονται σε πραγματικό χρόνο από τον διαχειριστή ή τους διαχειριστές της εφαρμογής.
- Να υποστηρίζει την επεξεργασία των δεδομένων των χρηστών με τεχνολογίες νέφους (Firebase)
- Να προσφέρει αυξημένη λειτουργικότητα με τον απλό και εύχρηστο σχεδιασμό του γραφικού περιβάλλοντος
- Και να εξετάσει πώς εφαρμογές σαν αυτή μπορούν να βοηθήσουν στην εκπαίδευση.

Η εργασία θέλει να αποτελέσει ένα πρότυπο για την αξιοποίηση συγχρόνων τεχνολογιών όπως είναι το Android και η Firebase στην εκπαίδευση, δίνοντας έμφαση στην αυτοαξιολόγηση και στη συνεχή ανατροφοδότηση του φοιτητή.

1.2 Αντικείμενο και σημασία του έργου

Αντικείμενο αυτής της εργασίας είναι η ανάπτυξη μιας πλήρους λειτουργικής εφαρμογής για συστήματα που βασίζονται στο Android. Η εφαρμογή προσφέρει στον φοιτητή ερωτήσεις πολλαπλής επιλογής για ποικιλία γνωστικών αντικειμένων που περιορίζονται μόνο από τους εκάστοτε διαχειριστές της εφαρμογής, δίνοντάς του τη δυνατότητα να αυτοαξιολογηθεί σε αυτά.

Σημαντικό πλεονέκτημα αποτελεί το γεγονός ότι οι διδάσκοντες και κάθε διαχειριστής της εφαρμογής μπορεί να εμπλουτίζει την εφαρμογή με περιεχόμενο για τους φοιτητές, χωρίς να απαιτείται εξειδικευμένη γνώση από μέρους του.

Στο τεχνολογικό επίπεδο γίνεται χρήση σύγχρονων τεχνολογιών. Χρησιμοποιείται η γλώσσα Kotlin και το Jetpack Compose καθώς και η υπηρεσία νέφους Firebase για την αποθήκευση δεδομένων, την αυθεντικοποίηση των χρηστών και τη διαχείριση του περιεχομένου. Με αυτό τον τρόπο έχουμε αυξημένη λειτουργικότητα, ταχύτητα και ασφάλεια καθώς και τη δυνατότητα μελλοντικών επεκτάσεων της εφαρμογής.

Συνολικά, η παρούσα διπλωματική εργασία προτείνει μια σύγχρονη και παιδαγωγική λύση που ενισχύει την αυτονομία των φοιτητών και την εμπλοκή τους στη μαθησιακή διαδικασία.

1.3 Μεθοδολογία προσέγγισης

Η μεθοδολογία που ακολουθήθηκε για την ανάπτυξη αυτής της εφαρμογής είχε τα εξής στάδια: Ανάλυση, σχεδίαση, υλοποίηση και αξιολόγηση.

Στο πρώτο στάδιο αναλύθηκαν οι απαιτήσεις της εφαρμογής λαμβάνοντας υπόψιν τόσο τις ανάγκες των φοιτητών ως τελικών χρηστών, όσο και των διδασκόντων και διαχειριστών του συστήματος για να ανανεώνουν το περιεχόμενο της εφαρμογής. Έχουμε δηλαδή δύο ρόλους: του φοιτητή, που αξιοποιεί το περιεχόμενο και του διαχειριστή, που δημιουργεί το περιεχόμενο.

Στο επόμενο στάδιο σχεδιάστηκε η αρχιτεκτονική της εφαρμογής. Στοχεύθηκε η δημοφιλής πλατφόρμα Android με τη χρήση της γλώσσας Kotlin και του Jetpack Compose για το γραφικό περιβάλλον της εφαρμογής. Σαν περιβάλλον ανάπτυξης επιλέχθηκε το Android Studio μιας και περιέχει όλα τα απαραίτητα εργαλεία για την υλοποίηση της εφαρμογής. Τέλος, από την

πλατφόρμα νέφους Firebase χρησιμοποιήθηκε η Real Time Database για την αποθήκευση των δεδομένων των χρηστών και του περιεχομένου της εφαρμογής και για τους διαχειριστές του συστήματος η υπηρεσία Firebase Authentication.

Στη φάση της υλοποίησης δόθηκε προσοχή σε ένα απλό και εύχρηστο περιβάλλον χρήσης. Η σχεδίαση των οθονών όπως η επιλογή γνωστικού αντικειμένου, η διεξαγωγή του τεστ, η προβολή του αποτελέσματος έγινε ώστε να ακολουθείται ο πιο φυσικός και άμεσος τρόπος για κάθε λειτουργία της εφαρμογής. Οι διαχειριστές του συστήματος έχουν το δικό τους περιβάλλον, στο οποίο μπορούν να χρησιμοποιήσουν για να δημιουργήσουν θέματα, υποκατηγορίες και ερωτήσεις, καθώς και να κάνουν όλα τα προηγούμενα με την εισαγωγή αρχείου JSON.

Τέλος, πραγματοποιήθηκαν δοκιμές της εφαρμογής, ώστε να διασφαλιστεί η ομαλή συμπεριφορά της σε όλα τα σενάρια χρήσης σε Android Emulator, καθώς και σε φυσικές συσκευές.

Η μεθοδολογία αυτή εξασφάλισε την εύκολη συντήρηση της εφαρμογής καθώς και τη δυνατότητα μελλοντικών επεκτάσεών της, σύμφωνα με τις εκπαιδευτικές και τεχνολογικές ανάγκες που θα προκύψουν.

1.4 Δομή της διπλωματικής εργασίας

Η εργασία είναι οργανωμένη σε κεφάλαια, καθένα από τα οποία αναπτύσσει με συστηματικό τρόπο ένα συγκεκριμένο μέρος της διαδικασίας που ακολουθήθηκε, από τον αρχικό σχεδιασμό μέχρι και την τελική αξιολόγηση της εφαρμογής. Η διάρθρωση επιλέχθηκε έτσι ώστε να διευκολύνει τον αναγνώστη στην κατανόηση της πορείας ανάπτυξης και των βασικών τεχνολογικών επιλογών που υλοποιήθηκαν.

Το Πρώτο Κεφάλαιο, στο οποίο παρουσιάστηκε η εισαγωγή στο αντικείμενο της εργασίας, οι λόγοι που οδήγησαν στην επιλογή του θέματος, καθώς και οι στόχοι που τέθηκαν εξαρχής. Εξηγείται επίσης η σημασία του έργου και η μεθοδολογική προσέγγιση που ακολουθήθηκε κατά την ανάπτυξή του.

Το Δεύτερο Κεφάλαιο παρουσιάζει τις βασικές θεωρητικές αρχές που σχετίζονται με την αυτοαξιολόγηση στη μαθησιακή διαδικασία και τον ρόλο της τεχνολογίας στην εκπαίδευση. Αναλύεται η σημασία της αυτορρυθμιζόμενης μάθησης, η συμβολή των ψηφιακών εργαλείων

στην ενίσχυση της συμμετοχής των εκπαιδευομένων και εξετάζονται υπάρχουσες πλατφόρμες και εφαρμογές αυτοαξιολόγησης. Το κεφάλαιο αυτό θέτει το υπόβαθρο πάνω στο οποίο στηρίχθηκε ο σχεδιασμός και η υλοποίηση της εφαρμογής που αναπτύχθηκε στην παρούσα εργασία.

Το Τρίτο Κεφάλαιο παραθέτει τις τεχνολογικές επιλογές (Kotlin, Jetpack Compose, Firebase) και την αρχιτεκτονική της εφαρμογής, με έμφαση στη modular δομή και την πλοήγηση μεταξύ των βασικών οθονών (Home, Quiz, Results, Leaderboard, Admin). Περιγράφεται η οργάνωση των δεδομένων και η ασύγχρονη επικοινωνία με το Firebase, καθώς και ο σχεδιασμός του UI βάσει Material Design και δηλωτικού Compose. Αναλύεται η ροή λειτουργίας (φόρτωση ερωτήσεων, υπολογισμός σκορ, αποθήκευση αποτελεσμάτων) και ο ρόλος του Admin Dashboard για δυναμική διαχείριση περιεχομένου. Τέλος, συνοψίζονται οι πρακτικές υλοποίησης, οι επιλογές σχεδίασης και οι δοκιμές που διασφαλίζουν σταθερότητα, επεκτασιμότητα και καλή εμπειρία χρήστη.

Το Τέταρτο Κεφάλαιο, στο οποίο γίνεται ανάλυση του κώδικα της εφαρμογής.

Το Πέμπτο Κεφάλαιο, στο οποίο παρουσιάζονται οι περιπτώσεις χρήσης της εφαρμογής με την επίδειξη των διάφορων οθονών της εφαρμογής.

Το Έκτο Κεφάλαιο, που παρουσιάζει τη συνολική αποτίμηση της εφαρμογής, τα συμπεράσματα που προέκυψαν από την ανάπτυξη και τη δοκιμή της, καθώς και την παιδαγωγική της αξία ως εργαλείο αυτοαξιολόγησης φοιτητών. Επισημαίνονται τα πλεονεκτήματα και οι περιορισμοί της παρούσας υλοποίησης, ενώ προτείνονται συγκεκριμένες κατευθύνσεις για μελλοντική εξέλιξη και εμπλουτισμό των λειτουργιών της εφαρμογής. Τέλος, διατυπώνεται ο επίλογος, ο οποίος συνοψίζει τη συμβολή της εργασίας στη σύγχρονη εκπαιδευτική διαδικασία.

Κεφάλαιο 2: Θεωρητικό Υπόβαθρο και Σχετική Έρευνα

2.1 Η έννοια της αυτοαξιολόγησης στη μαθησιακή διαδικασία

Η αυτοαξιολόγηση είναι μια ουσιαστική διαδικασία στην εκπαιδευτική πραγματικότητα. Ο φοιτητής μπορεί να ελέγξει τα δυνατά και αδύνατα σημεία του και να προβεί στις απαραίτητες ενέργειες για την εκπαιδευτική του πορεία, αποτελεί δηλαδή ένα εργαλείο αυτογνωσίας.

Η αυτοαξιολόγηση συμβάλλει σημαντικά στην μεταγνωστική ανάπτυξη του εκπαιδευόμενου. Τον ενθαρρύνει να στοχαστεί πάνω στη διαδικασία της μάθησης και όχι μόνο στα αποτελέσματά της. Του επιτρέπει να θέτει ρεαλιστικούς στόχους και να επιλέγει τις μεθόδους μάθησης. Η διαδικασία της μάθησης γίνεται συστηματική.

Σε πρακτικό επίπεδο, η αυτοαξιολόγηση συνδέεται στενά με τα εργαλεία ανατροφοδότησης. Ο εκπαιδευόμενος λαμβάνει άμεσα αποτελέσματα και καταλαβαίνει ποια γνώση έχει εμπεδωθεί. Είναι ιδιαίτερα χρήσιμη στην εξ αποστάσεως εκπαίδευση που υπάρχει περιορισμένη επαφή με τον διδάσκοντα, καθώς και στο e-learning.

Η αυτοαξιολόγηση βοηθάει στην αυτονομία των εκπαιδευόμενων. Αντί να εξαρτώνται από τον εκπαιδευτικό αποκτούν οι ίδιοι τα μέσα και την αυτοπεποίθηση να ελέγχουν μόνοι τους την πρόοδό τους. Αυτή η μετατόπιση αποτελεί θεμελιώδες χαρακτηριστικό της σύγχρονης, μαθητοκεντρικής παιδαγωγικής προσέγγισης. Η παρούσα εφαρμογή αξιοποιεί αυτή την φιλοσοφία.

2.2 Ο ρόλος της τεχνολογίας στη μαθησιακή διαδικασία

Η εξέλιξη της τεχνολογίας τα τελευταία χρόνια έχει αλλάξει σε μεγάλο βαθμό το πώς προσεγγίζεται η εκπαίδευση. Η μετάβαση από τις παραδοσιακές μορφές διδασκαλίας σε ψηφιακές και υβριδικές μορφές διδασκαλίας δεν είναι πλέον καινοτομία, αλλά αναγκαιότητα. Οι τεχνολογίες πληροφορίας και επικοινωνιών (ΤΠΕ) έχουν ανοίξει νέους δρόμους για την πρόσβαση στη γνώση, στη συνεργατική μάθηση και την εξατομικευμένη υποστήριξη των εκπαιδευόμενων.

Η τεχνολογία στην εκπαίδευση δεν είναι μόνο μέσο μετάδοσης πληροφοριών αλλά ένα εργαλείο που βοηθάει στη συμμετοχή, την αλληλεπίδραση και την ανατροφοδότηση των εκπαιδευομένων. Ο εκπαιδευόμενος αποκτά πλούσιο υλικό, συνεργάζεται με άλλους εκπαιδευομένους και παρακολουθεί το μάθημα ανεξάρτητα τόπου και χρόνου. Ο εκπαιδευτικός προσαρμόζει το μάθημα σύμφωνα με τις ανάγκες των εκπαιδευόμενων.

Ιδιαίτερα σημαντική είναι η συμβολή της τεχνολογίας στην αυτορρυθμιζόμενη μάθηση. Μέσα από εφαρμογές, πλατφόρμες και ψηφιακά εργαλεία αξιολόγησης, οι εκπαιδευόμενοι αναπτύσσουν δεξιότητες αυτοπαρακολούθησης και αυτοαξιολόγησης. Αυτό τους επιτρέπει να αναπτύξουν αίσθημα ευθύνης και να ελέγξουν την πορεία τους.

Τα κινητά τηλέφωνα και οι φορητές συσκευές παίζουν καθοριστικό ρόλο. Η κινητή μάθηση (mobile learning) καταργεί τους περιορισμούς τόπου και χρόνου στην εκπαίδευση. Η φορητότητα και η αμεσότητα των κινητών εφαρμογών προσφέρουν νέες ευκαιρίες για πιο βιωματική και πιο διαδραστική μάθηση.

Επιπλέον, οι τεχνολογίες υπολογιστικού νέφους (cloud), όπως το Firebase στην παρούσα εργασία, προσφέρουν αποθήκευση δεδομένων, διαχείριση χρηστών και συγχρονισμό σε πραγματικό χρόνο. Έτσι οι πληροφορίες είναι διαθέσιμες σε κάθε συσκευή χωρίς να χρειάζεται πολύπλοκη διαχείριση και εγκατάσταση.

Τέλος, αξίζει να σημειωθεί ότι αυτά τα εργαλεία δεν έχουν σκοπό την αντικατάσταση του εκπαιδευτικού, αλλά τη διευκόλυνση και τη συμπλήρωση του έργου του. Ο εκπαιδευτικός ασχολείται με την καθοδήγηση και την ενδυνάμωση των μαθητών και λιγότερο με τη μηχανική αξιολόγηση των γνώσεων.

Στο πλαίσιο αυτό, η εφαρμογή που αναπτύχθηκε ανήκει στον τομέα της αυτοαξιολόγησης. Ο χρήστης παίρνει μέρος σε σύντομα τεστ γνώσεων για διάφορα θεματικά αντικείμενα σε ένα εύχρηστο περιβάλλον. Ο εκπαιδευτικός έχει τη δυνατότητα ανανέωσης και προσθήκης περιεχομένου.

2.3 Πλατφόρμες και εφαρμογές αυτοαξιολόγησης

Η ραγδαία ανάπτυξη των ψηφιακών τεχνολογιών έχει δημιουργήσει πλήθος πλατφορμών και εργαλείων που υποστηρίζουν την αυτοαξιολόγηση. Δεν έχουν όλες την ίδια λειτουργικότητα και δεν απευθύνονται στο ίδιο κοινό. Έχουν όμως τον ίδιο σκοπό, που είναι η συμμετοχή των εκπαιδευομένων στη μαθησιακή διαδικασία καθώς και η δυνατότητα να μπορούν να ελέγξουν την πρόοδό τους με τρόπο άμεσο και κατανοητό.

Ανάμεσα στα πιο γνωστά περιβάλλοντα συγκαταλέγονται οι πλατφόρμες διαχείρισης μάθησης (Learning Management Systems – LMS), όπως το Moodle (https://moodle.org/), το Blackboard (https://www.anthology.com/products/teaching-and-learning/learning-effectiveness/blackboard) και το Canvas (https://www.instructure.com/canvas). Είναι πλατφόρμες δημιουργίας και διαχείρισης μαθημάτων και προσφέρουν και ασκήσεις αυτοαξιολόγησης. Είναι όμως δύσκολες στη χρήση και συνήθως απαιτείται υπολογιστής, κάτι το οποίο πολλές φορές δεν επιτρέπει τη φορητότητα.

Εκτός από τα LMS, έχουν εμφανιστεί και διαδραστικές εφαρμογές αξιολόγησης, όπως τα Kahoot! (https://kahoot.com/), Quizizz (https://wayground.com/?lng=en) και Socrative (https://www.socrative.com/), που επιτρέπουν τη δημιουργία κουίζ και την ανατροφοδότηση στους χρήστες. Προσπαθούν να δώσουν μια παιγνιώδη προσέγγιση στη μάθηση. Η χρήση τους είναι συχνά περιορισμένη μιας και απαιτούν πρόσβαση στο διαδίκτυο καθώς και εγγραφή σε εξωτερικές πλατφόρμες, κάτι το οποίο δεν είναι πάντα επιθυμητό ή εφικτό.

Υπάρχουν εφαρμογές που εστιάζουν στη μεμονωμένη εξάσκηση και αυτοβελτίωση χωρίς να στοχεύουν σε κάποιο οργανωμένο μάθημα. Τέτοιες εφαρμογές ενισχύουν την αυτονομία του χρήστη και του επιτρέπουν να επαναλάβει το τεστ όσες φορές θέλει. Αυτή η προσέγγιση είναι πιο ευέλικτη και ελκυστική για φοιτητές που θέλουν να μελετούν ανεξάρτητα ή να καλύπτουν συγκεκριμένα γνωστικά κενά.

Η παρούσα εφαρμογή ανήκει σε αυτή ακριβώς την κατηγορία. Ο χρήστης μπορεί να εκτελέσει ένα quiz στη θεματολογία που τον ενδιαφέρει με τη βοήθεια μιας οργανωμένης βάσης δεδομένων. Απαντά σε ερωτήσεις πολλαπλής επιλογής και βλέπει άμεσα το αποτέλεσμα της προσπάθειάς του.

Επιπλέον, μπορεί να συγκρίνει τις επιδόσεις του με άλλους χρήστες της εφαρμογής, μιας και τα αποτελέσματα αποθηκεύονται στην Firebase Realtime Database.

Ένα ακόμη χαρακτηριστικό που διαφοροποιεί αυτή την εφαρμογή από τις ήδη υπάρχουσες λύσεις είναι η δυνατότητα διαχείρισης του περιεχομένου από τον διαχειριστή. Σε ένα απλό περιβάλλον ο διαχειριστής μπορεί να προσθέτει θεματολογίες, ερωτήσεις και να διαμορφώνει το περιεχόμενο όπως θέλει.

Συνολικά, οι πλατφόρμες αυτοαξιολόγησης είναι πλέον αναπόσπαστο κομμάτι της εκπαιδευτικής διαδικασίας. Η ενσωμάτωσή τους σε κινητές συσκευές τις κάνει ιδανικά εργαλεία για τη μαθησιακή αυτονομία του φοιτητή. Η εφαρμογή που υλοποιήθηκε στο πλαίσιο αυτής της εργασίας στοχεύει να αξιοποιήσει όλα τα παραπάνω πλεονεκτήματα, προσφέροντας ένα απλό αλλά ολοκληρωμένο σύστημα αυτοαξιολόγησης προσαρμοσμένο στις ανάγκες της σύγχρονης εκπαιδευτικής πραγματικότητας.

Κεφάλαιο 3 – Σχεδίαση και ανάπτυξη της εφαρμογής

3.1 Επιλογή τεχνολογιών και εργαλείων ανάπτυξης

Η επιλογή των τεχνολογιών για την ανάπτυξη αυτής της εφαρμογής δεν έγινε τυχαία, αλλά με γνώμονα την ευχρηστία, τη σταθερότητα, την επεκτασιμότητα και τη συμβατότητα με τις σύγχρονες τάσεις ανάπτυξης λογισμικού για κινητές συσκευές. Στόχος ήταν μια εφαρμογή που να συνδυάζει τη λειτουργικότητα και την εκπαιδευτική αξία.

Η γλώσσα προγραμματισμού που επιλέχθηκε είναι η Kotlin, η οποία είναι μία από τις κύριες γλώσσες ανάπτυξης εφαρμογών για το λειτουργικό σύστημα Android. Η Kotlin προσφέρει σύγχρονα χαρακτηριστικά, όπως ασφάλεια τύπων (type safety), απλότητα στη σύνταξη και πλήρη συμβατότητα με την Java, επιτρέποντας τη συνύπαρξη παλαιότερου και νεότερου κώδικα. Επίσης, διευκολύνεται με αυτήν η συντήρηση και η επεκτασιμότητα της εφαρμογής για τυχόν μελλοντικές αναβαθμίσεις.

Για το γραφικό περιβάλλον χρησιμοποιήθηκε το Jetpack Compose, η σύγχρονη βιβλιοθήκη της Google για την κατασκευή διεπαφών χρήστη (UI). Το παραδοσιακό XML σύστημα θεωρείται πλέον ξεπερασμένο. Η σχεδίαση με το Jetpack Compose είναι πιο ευέλικτη και λιγότερο επιρρεπής σε λάθη. Προσφέρει λειτουργικότητα και αισθητική, κρίσιμα στοιχεία για την ανάπτυξη μιας εκπαιδευτικής εφαρμογής.

Για τη διαχείριση δεδομένων και τη συγχρονισμένη επικοινωνία της εφαρμογής με τον διακομιστή επιλέχθηκε το Firebase Realtime Database. Αναπτύσσεται από την Google και περιέχει ένα σύνολο χαρακτηριστικών απαραίτητων για την εφαρμογή μας, έτσι δεν χρειαζόμαστε δικούς μας διακομιστές. Αποθηκεύονται το περιεχόμενο της εφαρμογής, οι ερωτήσεις και τα δεδομένα των χρηστών και ανακτώνται σε πραγματικό χρόνο, χωρίς επιπλέον εγκαταστάσεις ή επανεκκινήσεις της εφαρμογής.

Επίσης, το Firebase προσφέρει δυνατότητα Authentication, κάτι που είναι απαραίτητο ειδικότερα για τους διαχειριστές της εφαρμογής. Μόνο πιστοποιημένοι χρήστες μπορούν να ελέγξουν το περιεχόμενο της εφαρμογής, κάτι που διασφαλίζει την απαραίτητη αξιοπιστία και ποιότητα του περιεχομένου.

Για τη συνολική ανάπτυξη χρησιμοποιήθηκε το Android Studio, το επίσημο περιβάλλον ανάπτυξης της Google για Android εφαρμογές. Το Android Studio περιέχει πληθώρα εργαλείων, όπως ο emulator μιας συσκευής Android και ενσωμάτωση με το Git. Η χρήση του συγκεκριμένου IDE συνέβαλε στην ομαλή υλοποίηση, τον εύκολο έλεγχο λειτουργικότητας και την αποτελεσματική παρακολούθηση της επικοινωνίας με το Firebase.

Ο συνδυασμός όλων των παραπάνω εργαλείων αποδείχθηκε ιδανικός για την ανάπτυξη της εκπαιδευτικής εφαρμογής που θέλαμε. Συνολικά, οι επιλογές αυτές επιτρέπουν την ανάπτυξη μιας εφαρμογής που είναι απλή στη χρήση, εύκολα επεκτάσιμη και τεχνολογικά σύγχρονη, ανταποκρινόμενη πλήρως στους στόχους της εργασίας.

3.2 Αρχιτεκτονική και βασικές λειτουργίες της εφαρμογής

Η εφαρμογή σχεδιάστηκε ώστε να παρέχει ένα απλό και λειτουργικό περιβάλλον αυτοαξιολόγησης, το οποίο μπορεί να προσαρμοστεί σε διαφορετικά γνωστικά αντικείμενα. Έχει αρθρωτή δομή (modular architecture), με κάθε οθόνη να επιτελεί σαφή λειτουργία. Η πλοήγηση μεταξύ των οθονών υλοποιήθηκε με το Jetpack Compose.

3.2.1 Δομή και πλοήγηση

Η εφαρμογή αποτελείται από πέντε βασικές οθόνες:

- Την αρχική οθόνη (Home Screen). Αυτή αποτελείται από μια λίστα με τις θεματικές ενότητες οι οποίες είναι διαθέσιμες στον χρήστη και ανακτώνται από το Firebase. Υπάρχει και η επιλογή για σύνδεση διαχειριστή. Εάν επιλέξουμε μια θεματική ενότητα, τότε μεταβαίνουμε στις υποκατηγορίες της συγκεκριμένης και επιλέγουμε αυτή που θέλουμε. Στις υποκατηγορίες υπάρχει και επιλογή επιστροφής στην αρχική οθόνη.

- Την οθόνη ερωτήσεων (Quiz Screen). Εδώ βλέπουμε τις ερωτήσεις της υποκατηγορίας που επιλέξαμε παραπάνω, οι οποίες είναι πολλαπλής επιλογής και απαντάμε ανάλογα. Προχωράμε στην επόμενη ερώτηση με ένα κουμπί, αφού έχουμε επιλέξει πρώτα την απάντησή μας. Οι ερωτήσεις γίνονται με τυχαία σειρά κάθε φορά που διεξάγεται το τεστ.
- Την οθόνη αποτελεσμάτων (Results Screen). Εδώ βλέπουμε τη βαθμολογία που συγκεντρώσαμε από το τεστ. Αποθηκεύεται το σκορ μας στο Firebase με μοναδικό αναγνωριστικό συσκευής. Υπάρχει επιλογή να μεταβούμε στο Leaderboard ή να επιστρέψουμε στην αρχική οθόνη.
- Την οθόνη κατάταξης (Leaderboard Screen). Εδώ υπάρχει μια κατάταξη ανάλογα με τη θεματική ενότητα και την υποκατηγορία, με όλα τα αποτελέσματα από άλλους χρήστες, ώστε να προσφέρεται ένας φιλικός ανταγωνισμός μεταξύ των χρηστών.
- Την οθόνη διαχειριστή (Admin Dashboard). Σε αυτή έχει πρόσβαση μόνο ο διαχειριστής ή οι διαχειριστές της εφαρμογής. Το Authentication γίνεται μέσω Firebase. Εδώ ο διαχειριστής μπορεί να προσθέτει θέματα, υποκατηγορίες και ερωτήσεις, και υπάρχει και η επιλογή να τα εισάγει μέσω αρχείου JSON.

Η πλοήγηση ανάμεσα στις οθόνες πραγματοποιείται μέσω του NavHost Controller, που παρέχεται από το Jetpack Compose. Με αυτόν τον τρόπο το σύστημα παραμένει σταθερό και εύκολα επεκτάσιμο, ενώ κάθε οθόνη έχει τη δική της λογική και τα δικά της δεδομένα.

3.2.2 Επικοινωνία με το Firebase

Η εφαρμογή χρησιμοποιεί το Firebase Realtime Database για την αποθήκευση και ανάκτηση δεδομένων. Επίσης, χρησιμοποιεί το Firebase Authentication για να ελέγχει τα δικαιώματα των διαχειριστών. Τα δεδομένα οργανώνονται ιεραρχικά σε κόμβους (nodes). Αυτοί περιλαμβάνουν τη θεματολογία, τις υποκατηγορίες και τις ερωτήσεις. Αυτό κάνει το σύστημα εύκολα επεκτάσιμο με την προσθήκη νέου περιεχομένου ή την τροποποίηση του παλιού.

Η επικοινωνία της εφαρμογής με το Firebase γίνεται ασύγχρονα, μέσω των μεθόδων `addOnSuccessListener` και `addOnFailureListener`. Αυτό επιτρέπει στην εφαρμογή να

παραμένει λειτουργική ακόμα και όταν περιμένει απάντηση από τον διακομιστή. Ταυτόχρονα, έχουμε ενημέρωση σε πραγματικό χρόνο του περιεχομένου χωρίς την ανάγκη επανεκκινήσεων.

3.2.3 Δεδομένα και ροή πληροφορίας

Η ροή των δεδομένων μπορεί να συνοψιστεί ως εξής:

- Επιλέγουμε μάθημα και στη συνέχεια υποκατηγορία από την Home Screen.
- Μεταβαίνουμε στη Quiz Screen, όπου διεξάγονται οι ερωτήσεις, τις οποίες αντλούμε από το Firebase.
- Όταν το τεστ ολοκληρωθεί, εμφανίζεται η Results Screen και αποθηκεύεται το σκορ στο Firebase.
- Μπορούμε κατόπιν να μεταβούμε στη Leaderboard Screen και να δούμε το αποτέλεσμά μας σε σύγκριση με άλλους χρήστες.
- Οποιαδήποτε στιγμή μπορούμε να μεταβούμε στο Admin Dashboard για την προσθήκη ή τροποποίηση περιεχομένου.

Η αρχιτεκτονική αυτή, διαθέτοντας ασύγχρονη επικοινωνία και άμεση ενημέρωση, λειτουργεί απρόσκοπτα χωρίς επιπλοκές και παρέχει στον τελικό χρήστη μια ευχάριστη εμπειρία.

3.3 Σχεδίαση διεπαφής χρήστη (UI Design)

Η σχεδίαση της διεπαφής είναι κρίσιμο σημείο της εφαρμογής. Δόθηκε έμφαση στην απλότητα, ώστε να μπορεί οποιοσδήποτε να τη χρησιμοποιήσει χωρίς τεχνική εμπειρία.
Βασίστηκε στις αρχές του Material Design της Google, που επιτρέπει ευδιάκριτο σχεδιασμό των στοιχείων της εφαρμογής με έμφαση στο περιεχόμενο. Έτσι, η εφαρμογή διατηρεί μια καθαρή και λειτουργική αισθητική, με λίγα χρώματα, ευδιάκριτα κουμπιά και επαρκή απόσταση μεταξύ των στοιχείων, ώστε να μπορεί να χρησιμοποιηθεί άνετα σε συσκευές αφής.

3.3.1 Εργαλεία και τεχνολογίες σχεδίασης

Για την υλοποίηση της διεπαφής χρησιμοποιήθηκε το Jetpack Compose, το οποίο επιτρέπει τον δηλωτικό τρόπο ανάπτυξης (declarative UI). Αυτή η προσέγγιση καθιστά τη σχεδίαση πιο καθαρή, ευέλικτη και ευκολότερα συντηρήσιμη σε σχέση με τις παραδοσιακές μεθόδους ανάπτυξης μέσω XML.

Η χρήση του Compose επιτρέπει την επαναχρησιμοποίηση στοιχείων της εφαρμογής, όπως είναι τα κουμπιά απαντήσεων και οι κάρτες επιλογής.

Κάθε στοιχείο της εφαρμογής σχεδιάστηκε ως ανεξάρτητο σύνθετο (composable), κάτι που μειώνει σημαντικά τον πλεονάζοντα κώδικα.

Παράλληλα, επιτρέπεται η μελλοντική επέκταση της εφαρμογής χωρίς την ανάγκη επανασχεδιασμού της.

3.3.2 Οπτική ταυτότητα και χρωματική παλέτα

Η οπτική ταυτότητα της εφαρμογής αξιοποιεί τη χρωματική παλέτα του Material Design 3, που βασίζεται σε ήπιες χρωματικές αποχρώσεις. Χρησιμοποιήθηκε ως κύριο χρώμα ένα απαλό μπλε που αποπνέει ηρεμία. Ως δευτερεύοντα χρώματα χρησιμοποιήθηκαν αποχρώσεις του λευκού και του γκρι.

Δόθηκε έμφαση στην αναγνωσιμότητα των γραμμάτων με τις κατάλληλες γραμματοσειρές (όπως η Roboto), και δόθηκε προσοχή ώστε το κείμενο να είναι ευδιάκριτο ανεξάρτητα από το μέγεθος της συσκευής που θα χρησιμοποιηθεί.

3.3.3 Διάταξη και αλληλεπίδραση

Η δομή κάθε οθόνης σχεδιάστηκε ώστε να καθοδηγεί φυσικά τον χρήστη στη ροή της εφαρμογής. Η Home Screen δίνει έμφαση στις επιλογές μαθημάτων και υποκατηγοριών. Η Quiz Screen παρουσιάζει ευδιάκριτα τις ερωτήσεις μία προς μία, ενώ τα κουμπιά των απαντήσεων είναι επίσης ευδιάκριτα. Η Results Screen χρησιμοποιεί απλή γλώσσα με καθαρό κείμενο και διαθέτει δύο βασικά κουμπιά: τις επιλογές να μεταβεί ο χρήστης στην Home Screen ή στη Leaderboard Screen. Η Leaderboard Screen οργανώνει τις επιδόσεις των χρηστών με ευδιάκριτη παρουσίαση και διαφορετικό χρωματισμό για τον παρόντα χρήστη. Τέλος, στην Admin Dashboard η διεπαφή είναι πιο λειτουργική, με πεδία εισαγωγής και κουμπιά επεξεργασίας.

3.3.4 Προσαρμοστικότητα και εμπειρία χρήστη

Η διεπαφή σχεδιάστηκε ώστε να είναι πλήρως προσαρμόσιμη (responsive). Το Jetpack Compose φροντίζει ώστε να τηρούνται οι αναλογίες, είτε η εφαρμογή εκτελείται σε κινητό είτε σε ταμπλέτα. Επιπλέον, η εμπειρία χρήσης (UX) ενισχύεται από την απλότητα και την προβλεψιμότητα των αλληλεπιδράσεων. Ο χρήστης γνωρίζει ανά πάσα στιγμή πού βρίσκεται και τι πρέπει να κάνει, χωρίς περιττά γραφικά. Η συνολική προσέγγιση ακολουθεί τη λογική του «λιγότερο είναι περισσότερο».

Η επιλογή του Jetpack Compose, η καθαρή αισθητική του Material Design και η επικέντρωση στην απλότητα δημιούργησαν ένα περιβάλλον φιλικό, προσιτό και αποδοτικό.

.

3.4 Υλοποίηση και λειτουργικότητα της εφαρμογής

Η υλοποίηση της εφαρμογής πραγματοποιήθηκε μέσα από τα στάδια του προγραμματισμού, της δοκιμής και της αναθεώρησης. Βασική επιδίωξη ήταν η δημιουργία ενός εύχρηστου συστήματος που να μην απαιτεί τεχνικές γνώσεις από την πλευρά του χρήστη.

Η ανάπτυξη ακολούθησε τη λογική της τμηματικής υλοποίησης (modular development). Κάθε βασικό στοιχείο της εφαρμογής δοκιμάστηκε ανεξάρτητα πριν ενσωματωθεί στο τελικό έργο.

3.4.1 Δομή του κώδικα και οργάνωση αρχείων

Ο κώδικας οργανώθηκε σε ξεχωριστά αρχεία ανάλογα με τη λειτουργικότητα κάθε οθόνης.

Για παράδειγμα, υπάρχουν αρχεία όπως HomeScreen.kt, QuizScreen.kt, ResultsScreen.kt, LeaderboardScreen.kt και AdminDashboardScreen.kt, τα οποία αντιστοιχούν στα συγκεκριμένα τμήματα της εφαρμογής. Αυτό εξασφαλίζει ευκολία στη συντήρηση της εφαρμογής και κατανόηση από άλλους προγραμματιστές για την περαιτέρω ανάπτυξή της.

Η πλοήγηση μεταξύ των οθονών υλοποιείται μέσω του NavHost Controller του Jetpack Compose. Έτσι, η μετάβαση, για παράδειγμα, από την αρχική οθόνη στο τεστ γίνεται ομαλά.

3.4.2 Ανάκτηση και αποθήκευση δεδομένων

Η επικοινωνία με το Realtime Firebase Database αποτελεί τον τρόπο με τον οποίο αποθηκεύονται και ανακτώνται τα δεδομένα. Στην αρχή φορτώνονται οι κατηγορίες και οι υποκατηγορίες

δυναμικά και κατόπιν οι αντίστοιχες ερωτήσεις. Ο διαχειριστής μπορεί οποιαδήποτε στιγμή να ενημερώσει ή να προσθέσει περιεχόμενο, το οποίο γίνεται άμεσα διαθέσιμο στους χρήστες χωρίς επανεκκίνηση της εφαρμογής.

Κάθε φορά που ολοκληρώνεται ένα τεστ, αποθηκεύεται το σκορ του χρήστη με ένα αναγνωριστικό συσκευής και ημερομηνία, ώστε να είναι διαθέσιμο στο Leaderboard. Δεν χρειάζεται να κάνει ο χρήστης χειροκίνητη ανανέωση· τα αποτελέσματα είναι διαθέσιμα δυναμικά.

3.4.3 Οθόνη Quiz και λογική υπολογισμού σκορ

Η οθόνη του τεστ (QuizScreen.kt) αποτελεί ένα από τα πιο σημαντικά τμήματα της εφαρμογής. Κάθε ερώτηση εμφανίζεται ξεχωριστά και οι απαντήσεις προβάλλονται με τη μορφή κουμπιών που σχεδιάζονται δυναμικά. Οι απαντήσεις ανακατεύονται κάθε φορά που διεξάγεται το τεστ, ώστε να αποφεύγεται η απομνημόνευσή τους.

Κάθε φορά που ο χρήστης δίνει μια απάντηση και μεταβαίνει στην επόμενη ερώτηση, η εφαρμογή ελέγχει εάν είναι σωστή και, ανάλογα, αυξάνει το σκορ κατά μία μονάδα.

3.4.4 Αποθήκευση και ανάκτηση αποτελεσμάτων

Τα αποτελέσματα κάθε χρήστη αποθηκεύονται στο Firebase κάτω από τη δομή:

scores → <device_id> → <subject> → <subcategory> → entries

Η επιλογή αυτής της δομής διευκολύνει την ομαδοποίηση ανά χρήστη και θεματική ενότητα. Κατά την εμφάνιση του Leaderboard υπολογίζεται η μέγιστη βαθμολογία κάθε χρήστη. Εάν ο χρήστης δεν ανήκει στους δέκα πρώτους, εμφανίζεται στο τέλος της λίστας μια ένδειξη με το “your score:” και τη βαθμολογία του, ώστε να μη χάνεται η συνοχή.

3.4.5 Ρόλος και λειτουργίες του Διαχειριστή

Η εφαρμογή διαθέτει ειδική ενότητα για τον διαχειριστή που δεν είναι διαθέσιμη στους απλούς χρήστες. Μέσα από το Firebase Authentication κάνει σύνδεση με τον λογαριασμό που του έχει οριστεί. Υλοποιείται από το αρχείο AdminDashboardScreen.kt.

Μέσα από τον πίνακα ελέγχου, ο διαχειριστής μπορεί:

- να δημιουργήσει νέα μαθήματα και υποκατηγορίες,
- να προσθέσει ή να διαγράψει ερωτήσεις,
- να επεξεργαστεί υπάρχον περιεχόμενο,

- ή να εισαγάγει μαζικά δεδομένα μέσω αρχείου JSON.

Με την τελευταία επιλογή ο διαχειριστής μπορεί με μία κίνηση να εισαγάγει μεγάλο όγκο περιεχομένου με μια ενέργεια, χωρίς να διαγράφεται το παλιό περιεχόμενο.

3.4.6 Έλεγχος, δοκιμές και βελτιστοποίηση

Κατά τη διάρκεια της ανάπτυξης πραγματοποιήθηκαν πολλές δοκιμές τόσο σε εξομοιωτές Android (Android emulator) όσο και σε πραγματικές συσκευές. Χρησιμοποιήθηκαν διάφορες εκδόσεις Android καθώς και διαφορετικά μεγέθη συσκευών. Ιδιαίτερη έμφαση δόθηκε στη σταθερότητα της εφαρμογής σε διάφορες συνθήκες, όπως αποσυνδέσεις από το δίκτυο και η επικοινωνία με το Firebase.

Επιπλέον, δοκιμάστηκαν σενάρια όπως:

- αποσύνδεση από το διαδίκτυο κατά τη διάρκεια του τεστ,
- ταυτόχρονη αποθήκευση αποτελεσμάτων από διαφορετικούς χρήστες,
- και ενημέρωση περιεχομένου από τον διαχειριστή ενώ άλλοι χρήστες χρησιμοποιούν την εφαρμογή.

Η εφαρμογή αντέδρασε με σταθερότητα, εμφανίζοντας τα αντίστοιχα ενημερωτικά μηνύματα.

Η συνολική υλοποίηση σχεδιάστηκε με καθαρό και επεκτάσιμο κώδικα, ώστε να μπορεί να συντηρηθεί και να αναπτυχθεί περισσότερο στο μέλλον.

3.5 Αξιολόγηση και δοκιμές της εφαρμογής

Πραγματοποιήθηκε αξιολόγηση της εφαρμογής, ώστε να διαπιστωθεί ο βαθμός λειτουργικότητας και η ευκολία χρήσης.
Έγιναν δοκιμές τόσο από τον δημιουργό της εφαρμογής όσο και από φοιτητές, το κοινό στο οποίο απευθύνεται κυρίως η εφαρμογή. Με αυτόν τον τρόπο αποτιμήθηκε τόσο η τεχνική συμπεριφορά όσο και η εμπειρία του χρήστη (User Experience).

3.5.1 Μεθοδολογία αξιολόγησης

Η μεθοδολογία αξιολόγησης οργανώθηκε σε δύο φάσεις. Στην πρώτη πραγματοποιήθηκαν δοκιμές λειτουργικότητας (functional testing), όπου ελέγχθηκε η συνεργασία των διαφορετικών ενοτήτων της εφαρμογής. Η δεύτερη ήταν αξιολόγηση χρηστικότητας (Usability Testing) με τη συμμετοχή χρηστών που αλληλεπίδρασαν με την εφαρμογή και καταγράφηκαν οι εντυπώσεις τους.

Οι δοκιμές βασίστηκαν σε σενάρια χρήσης, όπως ο χρήστης να καλείται να πάρει μέρος σε ένα τεστ και να δει τα αποτελέσματά του και την κατάταξή του. Καταγράφηκαν οι ενέργειές τους και πιθανές δυσκολίες ή καθυστερήσεις που αντιμετώπισαν.

3.5.2 Τεχνικές δοκιμές

Οι τεχνικές δοκιμές επικεντρώθηκαν στη σταθερότητα της εφαρμογής, στη σωστή επικοινωνία με το Firebase και στη συμβατότητα με διαφορετικές εκδόσεις Android. Χρησιμοποιήθηκαν εξομοιωτές Android καθώς και πραγματικές συσκευές με διαφορετικό μέγεθος και ανάλυση.

Ελέγχθηκαν επιμέρους λειτουργίες, όπως:

- η φόρτωση μαθημάτων και υποκατηγοριών από τη βάση,
- η εμφάνιση των ερωτήσεων με τυχαία σειρά,
- η αποθήκευση των αποτελεσμάτων ανά χρήστη,
- και η σωστή εμφάνιση της κατάταξης (leaderboard).

Κατά τη διάρκεια των δοκιμών δεν εντοπίστηκαν σφάλματα. Όταν η σύνδεση είναι αργή, εμφανίζεται η ένδειξη φόρτωσης (loading), η οποία αποτελεί αναμενόμενη και κατανοητή συμπεριφορά.

3.5.3 Δοκιμές χρηστικότητας

Για την αξιολόγηση της εμπειρίας χρήστη, δόθηκε η εφαρμογή σε δέκα φοιτητές διαφορετικών γνωστικών αντικειμένων. Η επιλογή έγινε ώστε να συμπεριλαμβάνει τόσο εξοικειωμένους με την τεχνολογία όσο και αρχάριους. Χωρίς καθοδήγηση χρησιμοποίησαν την εφαρμογή.

Μετά τη χρήση, οι συμμετέχοντες απάντησαν σε ένα σύντομο ερωτηματολόγιο που περιελάμβανε ερωτήσεις σχετικά με:

- την ευκολία κατανόησης της διεπαφής,

- την οργάνωση του περιεχομένου,
- την οπτική καθαρότητα,
- και τη συνολική εμπειρία χρήσης.

Η πλειονότητα των χρηστών θεώρησε την εφαρμογή απλή και ευχάριστη στη χρήση. Μικρές προτάσεις βελτίωσης αφορούσαν την αισθητική διαφοροποίηση των θεμάτων ή τη δυνατότητα εμφάνισης ενός σύντομου ιστορικού προηγούμενων προσπαθειών.

3.5.4 Ανάλυση αποτελεσμάτων

Η ανάλυση των δεδομένων από τις δοκιμές έδειξε ότι η εφαρμογή λειτουργεί αξιόπιστα και ικανοποιεί τους χρήστες ως προς την αλληλεπίδραση. Για ένα τεστ είκοσι ερωτήσεων, ο μέσος χρόνος που χρειάστηκε ήταν τέσσερα λεπτά, χρόνος που θεωρείται ικανοποιητικός για σύντομα τεστ. Οι χρήστες μπόρεσαν να χρησιμοποιήσουν την εφαρμογή χωρίς οδηγίες.

Η σταθερότητα του συστήματος ήταν υψηλή και το Firebase λειτούργησε αξιόπιστα. Οι λειτουργίες για προσθήκη και ενημέρωση περιεχομένου από τον διαχειριστή λειτούργησαν σωστά. Ιδιαίτερα χρήσιμη ήταν η εισαγωγή περιεχομένου από JSON αρχείο, καθώς μειώνει δραματικά τον χρόνο ενημέρωσης περιεχομένου.

3.5.5 Συμπεράσματα αξιολόγησης

Η διαδικασία αξιολόγησης επιβεβαίωσε ότι η εφαρμογή ανταποκρίνεται στους αρχικούς στόχους της. Προσφέρει ένα απλό και εύχρηστο εργαλείο αυτοαξιολόγησης. Η εμπειρία χρήστη είναι θετική και η τεχνική σταθερότητα ικανοποιητική. Οι παρατηρήσεις των χρηστών χρησιμοποιήθηκαν ως αφετηρία για μικρές διορθώσεις και για τον καθορισμό πιθανών μελλοντικών επεκτάσεων.

Κεφάλαιο 4 – Ανάλυση κώδικα της εφαρμογής

4.1 Κεντρική δραστηριότητα και πλοήγηση της εφαρμογής

```
package com.example.meta

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.runtime.Composable
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController
import com.example.meta.ui.theme.MetaTheme
import com.google.firebase.auth.auth
import com.google.firebase.Firebase
import android.util.Log
import com.google.firebase.auth.FirebaseAuth

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // ☑ Automatically sign in anonymously for Firebase security rules
```

```
        Firebase.auth.signInAnonymously()
            .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    println("☑ Firebase anonymous sign-in successful:
${Firebase.auth.currentUser?.uid}")
                } else {
                    println("❌ Firebase anonymous sign-in failed:
${task.exception?.message}")
                }
            }
        val auth = FirebaseAuth.getInstance()
        if (auth.currentUser == null) {
            auth.signInAnonymously()
                .addOnSuccessListener { Log.d("Auth", "Signed in
anonymously: ${it.user?.uid}") }
                .addOnFailureListener { e -> Log.e("Auth", "Anon sign-in
failed", e) }
        }
        setContent {
            MetaTheme {
                MetaApp()
            }
        }
    }
}

@Composable
fun MetaApp() {
    val navController = rememberNavController()

    NavHost(navController = navController, startDestination = "home") {
        // Home Screen
        composable("home") { HomeScreen(navController) }
        composable("login") { LoginScreen(navController) }

        // Quiz Screen with subject and subcategory
        composable("quiz/{subject}/{subcategory}") { backStackEntry ->
            val subject = backStackEntry.arguments?.getString("subject") ?:
""
            val subcategory =
backStackEntry.arguments?.getString("subcategory") ?: ""
            QuizScreen(navController = navController, subject = subject,
subcategory = subcategory)
        }

        // Results Screen with score, subject, and subcategory
        composable("results/{score}/{subject}/{subcategory}") {
backStackEntry ->
            val score =
backStackEntry.arguments?.getString("score")?.toInt() ?: 0
            val subject = backStackEntry.arguments?.getString("subject") ?:
""
            val subcategory =
backStackEntry.arguments?.getString("subcategory") ?: ""
            ResultsScreen(score = score, subject = subject, subcategory =
subcategory, navController = navController)
```

```
        }

        // Admin Dashboard
        composable("admin") { AdminDashboardScreen(navController) }

        // Leaderboard Screen with subject and subcategory
        composable("leaderboard/{subject}/{subcategory}") { backStackEntry -
>
            val subject = backStackEntry.arguments?.getString("subject") ?:
""
            val subcategory =
backStackEntry.arguments?.getString("subcategory") ?: ""
            LeaderboardScreen(subject = subject, subcategory = subcategory,
navController = navController)
        }
    }
}
```

1.Κώδικας του MainActivity.kt, ο οποίος αρχικοποιεί την εφαρμογή, ενεργοποιεί την ανώνυμη σύνδεση στο Firebase και ορίζει το σύστημα πλοήγησης του Edutest.

Η κλάση `MainActivity` αποτελεί το κύριο σημείο εισόδου της εφαρμογής Edutest στο περιβάλλον Android και λειτουργεί ως γέφυρα ανάμεσα στο «παραδοσιακό» lifecycle των δραστηριοτήτων (activities) και στο δηλωτικό περιβάλλον διεπαφής που προσφέρει το Jetpack Compose. Παράλληλα, σε αυτό το σημείο υλοποιείται και η αρχική σύνδεση του χρήστη στο Firebase μέσω ανώνυμης αυθεντικοποίησης, ώστε κάθε συσκευή να διαθέτει ένα μοναδικό αναγνωριστικό χρήστη (UID), το οποίο αξιοποιείται στη συνέχεια τόσο στους κανόνες ασφαλείας όσο και στην αποθήκευση σκορ.

Η `MainActivity` κληρονομεί από την `ComponentActivity` και υπερκαλύπτει τη μέθοδο `onCreate`. Στο εσωτερικό της μεθόδου αυτής, πριν ακόμη γίνει η απόδοση της διεπαφής χρήστη, εκτελείται η διαδικασία ανώνυμης εισόδου στο Firebase Authentication. Αρχικά, καλείται η `Firebase.auth.signInAnonymously()`, η οποία εκκινεί μια ασύγχρονη προσπάθεια σύνδεσης και συνδέεται με έναν `addOnCompleteListener`. Σε περίπτωση επιτυχούς σύνδεσης, τυπώνεται στο log/κονσόλα μήνυμα με το μοναδικό αναγνωριστικό χρήστη (`Firebase.auth.currentUser?.uid`), επιβεβαιώνοντας ότι η συσκευή διαθέτει πλέον ενεργό ανώνυμο χρήστη. Σε περίπτωση αποτυχίας, καταγράφεται μήνυμα σφάλματος με την αντίστοιχη εξαίρεση. Η λογική αυτή εξασφαλίζει ότι ακόμη και χρήστες που δεν διαθέτουν προσωπικό λογαριασμό (email/κωδικό) μπορούν να αλληλεπιδρούν με την εφαρμογή και να αποθηκεύουν τα αποτελέσματά τους στο cloud με συνέπεια, καθώς κάθε συσκευή αντιστοιχίζεται σε ένα UID.

Στη συνέχεια, δημιουργείται ένα δεύτερο αντικείμενο αυθεντικοποίησης μέσω της `FirebaseAuth.getInstance()`, το οποίο αποθηκεύεται στη μεταβλητή `auth`. Ακολουθεί ένας έλεγχος (`if (auth.currentUser == null)`), ώστε σε περίπτωση που –για οποιονδήποτε λόγο– δεν υπάρχει ήδη ενεργός χρήστης, να επιχειρείται εκ νέου ανώνυμη σύνδεση μέσω `auth.signInAnonymously()`. Η δεύτερη αυτή κλήση συνοδεύεται από δύο listener: στον `addOnSuccessListener` καταγράφεται στο log ότι η ανώνυμη σύνδεση ολοκληρώθηκε με επιτυχία, ενώ στον `addOnFailureListener` καταγράφεται τυχόν αποτυχία. Με αυτόν τον τρόπο, η εφαρμογή διασφαλίζει ότι, πρακτικά, δεν θα υπάρξει σενάριο χρήσης όπου η Edutest θα λειτουργεί χωρίς διαθέσιμο `currentUser`, κάτι που είναι κρίσιμο για τους κανόνες ασφαλείας του Firebase Realtime Database και για τη δομή αποθήκευσης των σκορ (π.χ. `scores/<uid>/...`).

Αφού ολοκληρωθεί η αρχική φάση αυθεντικοποίησης, καλείται η `setContent { ... }`, η οποία σηματοδοτεί τη μετάβαση στο δηλωτικό περιβάλλον του Jetpack Compose. Στο εσωτερικό της, εφαρμόζεται το θέμα της εφαρμογής (`MetaTheme`) και καλείται το composable `MetaApp()`, το οποίο αναλαμβάνει την οργάνωση της πλοήγησης και των βασικών οθονών.

Η συνάρτηση `MetaApp` ορίζει τη ρίζα της πλοήγησης της εφαρμογής, βασιζόμενη στο `NavHost` του Jetpack Compose. Αρχικά, δημιουργείται ένας `NavController` μέσω της `rememberNavController()`, ο οποίος κρατάει την τρέχουσα κατάσταση πλοήγησης (ποια οθόνη είναι ενεργή, τι παραμέτρους έχει κ.λπ.) και επιτρέπει την πλοήγηση μεταξύ των διαφορετικών composable οθονών. Ο `NavHost` δηλώνεται με `startDestination = "home"`, γεγονός που σημαίνει ότι η εφαρμογή ξεκινά πάντοτε από την αρχική οθόνη (HomeScreen).

Στο εσωτερικό του `NavHost` δηλώνονται όλες οι διαδρομές (routes) της εφαρμογής με τη μορφή `composable("route") { ... }`. Συγκεκριμένα:

- Η διαδρομή `"home"` αντιστοιχεί στην κλήση του `HomeScreen(navController)`, όπου ο χρήστης βλέπει τα διαθέσιμα μαθήματα και υποκατηγορίες, καθώς και τις επιλογές για πρόσβαση στο Admin Dashboard ή στη φόρμα σύνδεσης διαχειριστή.
- Η διαδρομή `"login"` αντιστοιχεί στην `LoginScreen(navController)`, η οποία επιτρέπει σε διαχειριστές να συνδεθούν με email/κωδικό μέσω Firebase Authentication και να αποκτήσουν δικαιώματα επεξεργασίας περιεχομένου.

- Η διαδρομή `"quiz/{subject}/{subcategory}"` ορίζει μια διαδρομή με παραμέτρους. Ο `NavHost` λαμβάνει ένα `backStackEntry`, από το οποίο ανακτώνται τα ορίσματα `subject` και `subcategory` μέσω `backStackEntry.arguments?.getString("subject")` και `backStackEntry.arguments?.getString("subcategory")`. Οι τιμές αυτές προωθούνται στη `QuizScreen`, ώστε η οθόνη του τεστ να φορτώσει τις αντιστοιχίσεις ερωτήσεις από το Firebase ανάλογα με το επιλεγμένο μάθημα και την επιμέρους ενότητα.
- Η διαδρομή `"results/{score}/{subject}/{subcategory}"` συνδέεται με την `ResultsScreen`. Από το `backStackEntry` ανακτάται το σκορ ως συμβολοσειρά και μετατρέπεται σε ακέραιο (`toInt()`), καθώς και ξανά το `subject` και το `subcategory`. Η `ResultsScreen` προβάλλει στον χρήστη τη βαθμολογία του και προσφέρει επιλογές επιστροφής στην αρχική οθόνη ή μετάβασης στον πίνακα κατάταξης.
- Η διαδρομή `"admin"` οδηγεί στο `AdminDashboardScreen(navController)`, το οποίο αποτελεί το κεντρικό περιβάλλον διαχείρισης θεμάτων, υποκατηγοριών και ερωτήσεων, καθώς και εισαγωγής δεδομένων από αρχεία JSON.
- Τέλος, η διαδρομή `"leaderboard/{subject}/{subcategory}"` συνδέεται με την `LeaderboardScreen`, όπου ανακτώνται και ταξινομούνται τα σκορ των χρηστών για το συγκεκριμένο μάθημα και υποκατηγορία. Και εδώ, οι παράμετροι `subject` και `subcategory` ανακτώνται δυναμικά από το `backStackEntry` και προωθούνται στον αντίστοιχο composable.

Η δομή αυτή καθιστά το `MetaApp` τον «κεντρικό χάρτη» πλοήγησης της εφαρμογής. Κάθε οθόνη είναι απομονωμένη σε δικό της composable, ενώ η ανταλλαγή δεδομένων (όπως subject, subcategory, score) γίνεται ρητά μέσω παραμέτρων στη διαδρομή. Αυτή η προσέγγιση έχει τρία βασικά πλεονεκτήματα:

1. **Καθαρή αρχιτεκτονική** – η MainActivity παραμένει ελαφριά, περιορισμένη στην αρχικοποίηση (auth + UI), ενώ η λογική πλοήγησης ζει αποκλειστικά στο επίπεδο του Compose.
2. **Επεκτασιμότητα** – η προσθήκη νέων οθονών ή παραμέτρων πλοήγησης γίνεται απλά με νέα `composable` routes, χωρίς ανάγκη μαζικών αλλαγών στον υπάρχον κώδικα.

3. **Συνοχή με το υπόλοιπο σύστημα** – η χρήση του ίδιου `NavController` σε όλες τις οθόνες επιτρέπει συνεπή πλοήγηση και προβλέψιμη συμπεριφορά, κάτι ιδιαίτερα σημαντικό για εκπαιδευτικές εφαρμογές όπου η καθαρότητα της ροής επηρεάζει άμεσα την εμπειρία μάθησης.

Συνοψίζοντας, ο συνδυασμός της `MainActivity` με το composable `MetaApp` υλοποιεί τον βασικό σκελετό της εφαρμογής: διασφαλίζει ότι κάθε χρήστης διαθέτει έγκυρη ανώνυμη ταυτότητα στο Firebase, εφαρμόζει το ενιαίο θέμα (MetaTheme) και ορίζει μια σαφή, επεκτάσιμη και παραμετροποιήσιμη αρχιτεκτονική πλοήγησης ανάμεσα στις βασικές λειτουργικές ενότητες της Edutest.

4.2 Αρχική οθόνη (HomeScreen) και επιλογή μαθήματος / υποκατηγορίας

Η αρχική οθόνη της εφαρμογής Edutest (HomeScreen) αποτελεί το βασικό σημείο εισόδου του χρήστη στο σύστημα αυτοαξιολόγησης και λειτουργεί ως κεντρικός κόμβος πλοήγησης προς τα διαθέσιμα μαθήματα, τις υποκατηγορίες τους και τον πίνακα διαχείρισης για τον διαχειριστή. Η υλοποίηση της οθόνης βασίζεται στο Jetpack Compose και συνδυάζει διαχείριση κατάστασης (state management), ασύγχρονη επικοινωνία με το Firebase Realtime Database και δυναμική διαμόρφωση του περιεχομένου ανάλογα με τα δεδομένα που είναι αποθηκευμένα στο νέφος.

```
var subjects by remember { mutableStateOf(listOf<String>()) }
var subcategories by remember { mutableStateOf(listOf<String>()) }
var selectedSubject by remember { mutableStateOf<String?>(null) }
var isLoading by remember { mutableStateOf(true)
```

2.Μεταβλητές κατάστασης στο HomeScreen.kt που χρησιμοποιούνται για την αποθήκευση των μαθημάτων, των υποκατηγοριών, της επιλεγμένης θεματικής ενότητας και της κατάστασης φόρτωσης δεδομένων από το Firebase.

Στο ανώτερο επίπεδο, η συνάρτηση `HomeScreen` ορίζει τέσσερις βασικές μεταβλητές κατάστασης: τη λίστα των μαθημάτων (`subjects`), τη λίστα των υποκατηγοριών (`subcategories`), το

επιλεγμένο μάθημα (`selectedSubject`) και μια λογική μεταβλητή που υποδεικνύει αν βρίσκεται σε εξέλιξη φόρτωση δεδομένων (`isLoading`). Οι μεταβλητές αυτές δηλώνονται μέσω του `remember { mutableStateOf(...) }`, ώστε να παρακολουθούνται αντιδραστικά από το Compose και να προκαλούν αυτόματη ανανέωση της διεπαφής όταν αλλάζει η τιμή τους. Με αυτόν τον τρόπο, η οθόνη παραμένει σύμφωνη με την τρέχουσα κατάσταση των δεδομένων, χωρίς να απαιτείται χειροκίνητη ενημέρωση της εμφάνισης.

```
LaunchedEffect(Unit) {
    val subjectsRef =
FirebaseDatabase.getInstance().getReference("subjects")
    subjectsRef.get()
        .addOnSuccessListener { snapshot ->
            subjects = snapshot.children.mapNotNull { it.value?.toString() }
            isLoading = false
        }
        .addOnFailureListener { e ->
            Log.e("HomeScreen", "Error loading subjects", e)
            isLoading = false
        }
```

3.Ανάκτηση της λίστας μαθημάτων από το Firebase Realtime Database κατά την αρχικοποίηση της HomeScreen, με χρήση του μηχανισμού LaunchedEffect του Jetpack Compose.

Η αρχική φόρτωση των διαθέσιμων μαθημάτων γίνεται μέσα σε ένα μπλοκ `LaunchedEffect(Unit)`, το οποίο εκτελείται μία φορά κατά την πρώτη «σύνθεση» της οθόνης. Η εφαρμογή συνδέεται στο Firebase Realtime Database, ανακτά το περιεχόμενο του κόμβου `subjects` και μετατρέπει τα παιδιά του κόμβου σε λίστα συμβολοσειρών, μέσω της έκφρασης `snapshot.children.mapNotNull { it.value?.toString() }`. Σε περίπτωση επιτυχούς ανάκτησης, η μεταβλητή `subjects` ενημερώνεται με τα ονόματα των μαθημάτων και η ένδειξη φόρτωσης (`isLoading`) γίνεται `false`. Σε περίπτωση αποτυχίας, καταγράφεται μήνυμα σφάλματος μέσω `Log.e(...)` και η διεπαφή ενημερώνεται ώστε να πάψει να εμφανίζει την ένδειξη φόρτωσης. Η χρήση του `LaunchedEffect` σε συνδυασμό με τους listeners `addOnSuccessListener` και `addOnFailureListener` επιτρέπει την ασύγχρονη επικοινωνία με τον διακομιστή, χωρίς να μπλοκάρεται το UI.

```
Column(
    modifier = Modifier
        .fillMaxSize()
        .padding(16.dp),
    horizontalAlignment = Alignment.CenterHorizontally,
    verticalArrangement = Arrangement.spacedBy(8.dp)
) {
    Text("Choose a Subject", style =
MaterialTheme.typography.headlineMedium)

    if (isLoading) {
```

4.Ορισμός της βασικής διάταξης της HomeScreen με χρήση Column, συμπεριλαμβανομένης της επικεφαλίδας και του μηχανισμού ένδειξης φόρτωσης δεδομένων.

Η διάταξη της οθόνης οργανώνεται με ένα `Column` που καλύπτει όλη την επιφάνεια της οθόνης και διατηρεί σταθερά περιθώρια (`padding(16.dp)`), οριζόντια κεντρική στοίχιση και σταθερή κάθετη απόσταση μεταξύ των στοιχείων. Στο επάνω μέρος εμφανίζεται ένας τίτλος (“Choose a Subject”) με τυπογραφία `headlineMedium`, ο οποίος δηλώνει με σαφήνεια τον στόχο της οθόνης. Αν η φόρτωση βρίσκεται σε εξέλιξη (`isLoading == true`), προβάλλεται ένας `CircularProgressIndicator`, υποδεικνύοντας στον χρήστη ότι η εφαρμογή αντλεί δεδομένα

```
if (selectedSubject == null) {
    // Subject list
    LazyColumn(
        modifier = Modifier
            .weight(1f)
            .fillMaxWidth(),
        verticalArrangement = Arrangement.spacedBy(8.dp)
    ) {
        items(subjects) { subject ->
            Card(
                modifier = Modifier
                    .fillMaxWidth()
                    .clickable {
                        selectedSubject = subject

                        // Load subcategories for this subject (robust
folder detection)
                        val subRef = FirebaseDatabase.getInstance()
                            .getReference("questions")
                            .child(subject)

                        subRef.get()
                            .addOnSuccessListener { snap ->
                                // Debug: list direct children
                                val childKeys = snap.children.mapNotNull {
it.key }
```

```
                                Log.d("HomeScreen", "children under $subject
= $childKeys")

                                // A child is a folder if ANY of its
grandchildren looks like a question (has questionText)
                                val folderKeys = snap.children.mapNotNull {
child ->
                                    val hasGrandchildQuestion =
child.children.any { grandChild ->

grandChild.child("questionText").exists()
                                    }
                                    Log.d("HomeScreen", "child ${child.key}
hasGrandchildQuestion=$hasGrandchildQuestion")
                                    if (hasGrandchildQuestion) child.key
else null
                                }

                                // A child is a leaf if ITSELF looks like a
question
                                val hasDirectLeaves = snap.children.any {
it.child("questionText").exists() }
                                Log.d("HomeScreen",
"hasDirectLeaves=$hasDirectLeaves, folderKeys=$folderKeys")

                                subcategories = when {
                                    folderKeys.isNotEmpty() -> folderKeys
                                    hasDirectLeaves -> listOf("General")
                                    else -> emptyList()
                                }
                            }
                            .addOnFailureListener { e ->
                                Log.e("HomeScreen", "Failed to load
subcategories", e)
                                subcategories = emptyList()
                            }

                    },
                elevation = CardDefaults.cardElevation(4.dp)
            ) {
                Text(subject, modifier = Modifier.padding(16.dp))
            }
        }
    }
```

5.Δυναμική προβολή μαθημάτων στην HomeScreen και αυτόματη ανίχνευση υποκατηγοριών από τη δομή του Firebase Realtime Database.

Όταν τα δεδομένα έχουν φορτωθεί και δεν έχει επιλεγεί ακόμη κάποιο μάθημα (`selectedSubject == null`), η οθόνη παρουσιάζει τη λίστα των διαθέσιμων μαθημάτων μέσω ενός `LazyColumn`. Κάθε μάθημα αποδίδεται μέσα σε μία `Card` η οποία είναι πλήρως κλικ-αριστή (`Modifier.clickable { ... }`). Με το πάτημα της κάρτας, ενημερώνεται η μεταβλητή

`selectedSubject` με το όνομα του επιλεγμένου μαθήματος και ενεργοποιείται η διαδικασία φόρτωσης των αντίστοιχων υποκατηγοριών από τη βάση δεδομένων.

Η ανάκτηση των υποκατηγοριών βασίζεται στο περιεχόμενο του κόμβου `questions/<subject>`. Η εφαρμογή ανακτά όλα τα παιδιά του συγκεκριμένου κόμβου (π.χ. Algebra, Geometry ή μεμονωμένες ερωτήσεις) και εφαρμόζει μια λογική διάκρισης ανάμεσα σε «φακέλους» (θεματικές υποενότητες) και «φύλλα» (άμεσες εγγραφές ερωτήσεων). Για κάθε παιδί, εξετάζονται τα «εγγόνια» του (grandchildren) και ελέγχεται εάν κάποιο από αυτά περιέχει πεδίο `questionText`. Αν υπάρχει τέτοιο εγγόνι, το αντίστοιχο παιδί θεωρείται φάκελος υποκατηγορίας και το κλειδί του προστίθεται στη λίστα `folderKeys`. Παράλληλα, ελέγχεται αν υπάρχουν «άμεσα» φύλλα, δηλαδή αν κάποιο από τα παιδιά του κόμβου έχει το ίδιο το πεδίο `questionText`. Στην περίπτωση αυτή, η εφαρμογή καταγράφει ότι υπάρχουν άμεσες ερωτήσεις (`hasDirectLeaves = true`) και, εφόσον δεν έχουν εντοπιστεί υποκατηγορίες, δημιουργεί μια εικονική υποκατηγορία με την ονομασία “General”. Η λογική αυτή επιτρέπει στην εφαρμογή να προσαρμόζεται σε δύο διαφορετικά στυλ δομής δεδομένων: είτε οργανωμένο σε υποφακέλους (π.χ. “Java”, “Python”), είτε με ερωτήσεις απευθείας κάτω από το μάθημα.

```
Text(
    "Select a Subcategory for $selectedSubject",
    style = MaterialTheme.typography.titleMedium
)

LazyColumn(
    modifier = Modifier
        .weight(1f)
        .fillMaxWidth(),
    verticalArrangement = Arrangement.spacedBy(8.dp)
) {
    items(subcategories) { sub ->
        Card(
            modifier = Modifier
                .fillMaxWidth()
                .clickable {
                    // Navigate to QuizScreen with both subject &
subcategory
                    navController.navigate("quiz/$selectedSubject/$sub")
                },
            elevation = CardDefaults.cardElevation(4.dp)
        ) {
            Text(sub, modifier = Modifier.padding(16.dp))
        }
    }
}

Spacer(modifier = Modifier.height(16.dp))
```

6.Υλοποίηση επιλογής υποκατηγορίας και δρομολόγηση της εφαρμογής με παραμέτρους μαθήματος και ενότητας

Αν έχει ήδη επιλεγεί ένα μάθημα και έχουν φορτωθεί οι σχετικές υποκατηγορίες (`selectedSubject != null`), η HomeScreen προβάλλει στη θέση της λίστας μαθημάτων μία νέα `LazyColumn` με τις υποκατηγορίες του επιλεγμένου μαθήματος. Στην κορυφή της ενότητας εμφανίζεται πληροφοριακό κείμενο του τύπου “Select a Subcategory for <subject>”, ώστε να γίνεται σαφές σε ποιο μάθημα αναφέρεται η τρέχουσα λίστα. Κάθε υποκατηγορία εμφανίζεται και πάλι ως κλικ-αριστή κάρτα και, όταν ο χρήστης την επιλέξει, ενεργοποιείται η πλοήγηση προς την `QuizScreen` με τη χρήση της διαδρομής `quiz/<subject>/<subcategory>`. Με αυτόν τον τρόπο, η QuizScreen λαμβάνει ως παραμέτρους τόσο το μάθημα όσο και την υποκατηγορία, ώστε να ανακτήσει τις αντίστοιχες ερωτήσεις από το Firebase.

```
Button(
    onClick = { selectedSubject = null },
    modifier = Modifier.fillMaxWidth()
) {
    Text("Back to Subjects")
}
```

7.Κουμπί επαναφοράς της κατάστασης επιλογής, με επιστροφή στη λίστα μαθημάτων.

Για την επιστροφή από το επίπεδο υποκατηγορίας στο επίπεδο επιλογής μαθήματος, παρέχεται ένα κουμπί “Back to Subjects”, το οποίο επαναφέρει τη μεταβλητή `selectedSubject` σε `null`. Αυτό έχει ως αποτέλεσμα την επανεμφάνιση της λίστας μαθημάτων, χωρίς να απαιτείται νέα ανάκτηση από τη βάση, αφού τα στοιχεία `subjects` παραμένουν ήδη αποθηκευμένα στην κατάσταση της οθόνης.

```
Val isAdmin = rememberIsAdmin()

    Spacer(modifier = Modifier.height(16.dp))

    if (isAdmin) {
        Button(
            onClick = { navController.navigate("admin") },
            modifier = Modifier.fillMaxWidth()
        ) { Text("Admin Dashboard") }

        TextButton(
            onClick = {
com.google.firebase.auth.FirebaseAuth.getInstance().signOut() },
            modifier = Modifier.fillMaxWidth()
        ) { Text("Sign Out") }
    } else {
        Button(
            onClick = { navController.navigate("login") },
            modifier = Modifier.fillMaxWidth()
        ) { Text("Admin Login") }
    }
}
```

8.Δυναμική προσαρμογή της διεπαφής ανάλογα με τον ρόλο του χρήστη (διαχειριστής ή απλός χρήστης).

Στο κάτω μέρος της HomeScreen ενσωματώνεται η λογική διαχείρισης δικαιωμάτων διαχειριστή. Η βοηθητική συνάρτηση `rememberIsAdmin()` χρησιμοποιεί το `FirebaseAuth` για να ανακτήσει το μοναδικό αναγνωριστικό του τρέχοντος χρήστη (`uid`) και στη συνέχεια ελέγχει, μέσω του κόμβου `admins/<uid>` στο Firebase Realtime Database, αν η τιμή που είναι αποθηκευμένη είναι `true`. Η τιμή αυτή αποθηκεύεται σε μια καταστασιακή μεταβλητή `isAdmin`, η οποία παρακολουθείται αντιδραστικά από το Compose. Αν ο χρήστης είναι καταγεγραμμένος ως διαχειριστής, εμφανίζεται ένα κουμπί “Admin Dashboard” που οδηγεί στην οθόνη διαχείρισης περιεχομένου, καθώς και ένα κουμπί “Sign Out” που επιτρέπει την αποσύνδεση από τον λογαριασμό. Αντίθετα, όταν ο χρήστης δεν είναι διαχειριστής ή δεν είναι συνδεδεμένος, παρουσιάζεται μόνο η επιλογή “Admin Login”, η οποία οδηγεί στην οθόνη σύνδεσης διαχειριστή.

Συνολικά, η HomeScreen συγκεντρώνει σε μία ενιαία οθόνη:

- τη δυναμική φόρτωση και απεικόνιση μαθημάτων από το Firebase,
- την έξυπνη ανίχνευση υποκατηγοριών ή γενικών ενοτήτων ανά μάθημα,
- την πλοήγηση προς τα κουίζ ανά subject και subcategory,
- και τη διαχείριση πρόσβασης διαχειριστή (admin vs. μη-authenticated χρήστης).

Με αυτόν τον τρόπο, λειτουργεί ως κεντρική διεπαφή οργάνωσης της μαθησιακής εμπειρίας, συνδέοντας τη δομή των δεδομένων στο νέφος με μια καθαρή, απλή και επεκτάσιμη διεπαφή χρήστη.

4.3 Ανάλυση της οθόνης Quiz (QuizScreen)

```
@Composable
fun QuizScreen(navController: NavHostController, subject: String,
subcategory: String) {

    var questions by remember { mutableStateOf(listOf<Question>()) }
    var currentQuestionIndex by remember { mutableStateOf(0) }
    var selectedAnswer by remember { mutableStateOf("") }
    var score by remember { mutableStateOf(0) }
    var isLoading by remember { mutableStateOf(true) }
```

9.Ορισμός μεταβλητών κατάστασης (state) για τη διαχείριση της ροής του κουίζ στην οθόνη Quiz.

Η οθόνη εκτέλεσης του τεστ (QuizScreen) αποτελεί τον πυρήνα της λειτουργικότητας της εφαρμογής Edutest, καθώς σε αυτήν υλοποιείται η διαδικασία αυτοαξιολόγησης του χρήστη μέσω ερωτήσεων πολλαπλής επιλογής. Η συνάρτηση `QuizScreen` υλοποιείται ως `@Composable` και δέχεται ως παραμέτρους τον `NavHostController`, καθώς και τα συμβολοσειρές `subject` και `subcategory`, οι οποίες ορίζουν το μάθημα και την υποκατηγορία που έχει επιλέξει ο φοιτητής από την αρχική οθόνη. Στο εσωτερικό της συνάρτησης χρησιμοποιούνται διάφορες μεταβλητές κατάστασης (`remember { mutableStateOf(...) }`), μέσω των οποίων αποθηκεύονται η λίστα των ερωτήσεων (`questions`), ο δείκτης της τρέχουσας ερώτησης (`currentQuestionIndex`), η επιλεγμένη απάντηση (`selectedAnswer`), το τρέχον σκορ (`score`) και μία βοηθητική σημαία φόρτωσης (`isLoading`). Με τον τρόπο αυτό, η διεπαφή συνδέεται άμεσα με την κατάσταση του

συστήματος: κάθε αλλαγή στα δεδομένα πυροδοτεί αυτόματη ανανέωση της οθόνης, χωρίς να απαιτείται ρητός χειρισμός.

```
LaunchedEffect(subject, subcategory) {
    val baseRef =
FirebaseDatabase.getInstance().getReference("questions").child(subject)
    val questionsRef = if (subcategory == "General") {
        // Leaves directly under the subject (q1, q2, ...)
        baseRef
    } else {
        // Nested under subcategory (Algebra/q1, Geometry/q1, ...)
        baseRef.child(subcategory)
    }

    questionsRef.get()
        .addOnSuccessListener { snapshot ->
            val loaded = snapshot.children.mapNotNull { child ->
                val text = child.child("questionText").value as? String
                val options = (child.child("options").value as?
List<*>)?.filterIsInstance<String>()
                val correct = child.child("correctAnswer").value as? String
                if (text != null && options != null && correct != null) {
                    Question(text, options, correct)
                } else null
            }.shuffled()

            questions = loaded
            isLoading = false
        }
        .addOnFailureListener { e ->
            Log.e("QuizScreen", "Error loading questions for
$subject/$subcategory", e)
            isLoading = false
        }
```

10.Ασύγχρονη φόρτωση ερωτήσεων από το Firebase Realtime Database με χρήση LaunchedEffect βάσει μαθήματος και υποκατηγορίας.

Η φόρτωση των ερωτήσεων από το Firebase Realtime Database πραγματοποιείται μέσα σε ένα μπλοκ `LaunchedEffect(subject, subcategory)`, το οποίο ενεργοποιείται κάθε φορά που μεταβάλλεται το μάθημα ή η υποκατηγορία. Αρχικά, δημιουργείται ένα βασικό reference προς τον κόμβο `questions/<subject>` και στη συνέχεια, ανάλογα με το αν η υποκατηγορία είναι "General" ή κάποια συγκεκριμένη θεματική, επιλέγεται είτε ο ίδιος ο κόμβος του μαθήματος είτε ο υποκόμβος της αντίστοιχης υποκατηγορίας. Η ανάκτηση των δεδομένων γίνεται ασύγχρονα μέσω της μεθόδου `get()`, και σε περίπτωση επιτυχίας η εφαρμογή διατρέχει τους απογόνους του snapshot, δημιουργώντας από κάθε παιδί ένα αντικείμενο `Question`, εφόσον υπάρχουν τα πεδία `questionText`, `options` και `correctAnswer`. Η λίστα των ερωτήσεων ανακατεύεται (`shuffled()`), ώστε η σειρά εμφάνισης να διαφέρει σε κάθε προσπάθεια, αποτρέποντας τη

μηχανική απομνημόνευση των σωστών θέσεων. Σε περίπτωση αποτυχίας, καταγράφεται μήνυμα σφάλματος μέσω `Log.e`, ενώ η σημαία `isLoading` απενεργοποιείται ώστε η διεπαφή να ενημερώσει τον χρήστη.

```
if (isLoading) {
    Box(modifier = Modifier.fillMaxSize(), contentAlignment =
Alignment.Center) {
        CircularProgressIndicator()
    }
    return
}

if (questions.isEmpty()) {
    Box(modifier = Modifier.fillMaxSize(), contentAlignment =
Alignment.Center) {
        Text("No questions available for $subject → $subcategory")
    }
    return
}
```

11.Διαχείριση καταστάσεων φόρτωσης και απουσίας δεδομένων κατά την εκτέλεση του κουίζ.

Η απεικόνιση της κατάστασης φόρτωσης και της διαθεσιμότητας ερωτήσεων υλοποιείται με σαφή και προβλέψιμο τρόπο. Όσο η μεταβλητή `isLoading` είναι `true`, η οθόνη προβάλλει έναν κυκλικό δείκτη προόδου (`CircularProgressIndicator`) στο κέντρο. Αν η λίστα `questions` παραμείνει κενή μετά τη φόρτωση, εμφανίζεται μήνυμα «No questions available for …», ενημερώνοντας τον χρήστη ότι δεν υπάρχουν διαθέσιμα δεδομένα για τον συγκεκριμένο συνδυασμό μαθήματος και υποκατηγορίας. Μόνο όταν υπάρχουν ερωτήσεις, η εφαρμογή συνεχίζει στη βασική ροή του κουίζ.

```
val currentQuestion = questions[currentQuestionIndex]
val optionsShuffled by remember(currentQuestionIndex) {
    mutableStateOf(currentQuestion.options.shuffled())
}

Column(
    modifier = Modifier
        .fillMaxSize()
        .padding(16.dp),
    horizontalAlignment = Alignment.CenterHorizontally,
    verticalArrangement = Arrangement.spacedBy(16.dp)
) {
    Text(currentQuestion.questionText, style =
MaterialTheme.typography.titleLarge)

    // Answers
    LazyColumn(
        verticalArrangement = Arrangement.spacedBy(8.dp),
        modifier = Modifier.weight(1f, fill = false) // keeps the
Submit/Next button visible
    ) {
        items(optionsShuffled) { option ->
            AnswerOption(
                option = option,
                isSelected = option == selectedAnswer,
                onClick = { selectedAnswer = option }
            )
        }
    }
```

12.Ανάκτηση της τρέχουσας ερώτησης και δυναμική εμφάνιση των απαντήσεων με τυχαία σειρά.

Για κάθε βήμα του τεστ, επιλέγεται η τρέχουσα ερώτηση μέσω του `currentQuestionIndex` και αποθηκεύεται στη μεταβλητή `currentQuestion`. Παράλληλα, οι απαντήσεις της ερώτησης ανακατεύονται εκ νέου σε τοπικό επίπεδο (`optionsShuffled`) με χρήση του `remember(currentQuestionIndex)`, ώστε οι επιλογές να εμφανίζονται κάθε φορά σε διαφορετική σειρά, αλλά να παραμένουν σταθερές όσο ο χρήστης βρίσκεται σε αυτήν τη συγκεκριμένη ερώτηση. Η διάταξη της οθόνης βασίζεται σε ένα `Column`, όπου στο πάνω μέρος εμφανίζεται το κείμενο της ερώτησης και ακολουθεί μια κάθετη λίστα (`LazyColumn`) με τα κουμπιά απαντήσεων. Κάθε απάντηση υλοποιείται μέσω της συνάρτησης `AnswerOption`, η οποία αποδίδει ένα `Button` πλήρους πλάτους. Ο οπτικός διαχωρισμός της επιλεγμένης επιλογής επιτυγχάνεται με αλλαγή του χρώματος φόντου ανάλογα με την τιμή του `isSelected`, ενισχύοντας τη σαφήνεια της αλληλεπίδρασης.

```
Button(
    onClick = {
        if (selectedAnswer == currentQuestion.correctAnswer) score++

        if (currentQuestionIndex < questions.size - 1) {
            currentQuestionIndex++
            selectedAnswer = ""
        } else {
            // Finished quiz — save score per subject + subcategory using
Firebase UID
            val uid = FirebaseAuth.getInstance().currentUser?.uid
            if (uid == null) {
                Log.e("QuizScreen", "No authenticated user; cannot write
score")
                // Navigate anyway so user isn't blocked

navController.navigate("results/$score/$subject/$subcategory")
            } else {
                val scoresRef =
FirebaseDatabase.getInstance().getReference("scores")
                val scoreEntry = mapOf(
                    "score" to score,
                    "timestamp" to System.currentTimeMillis()
                )
                scoresRef.child(uid).child(subject).child(subcategory)
                    .push()
                    .setValue(scoreEntry)
                    .addOnSuccessListener {
                        Log.d(
                            "QuizScreen",
                            "Score saved OK: user=$uid subject=$subject
sub=$subcategory score=$score"
                        )

navController.navigate("results/$score/$subject/$subcategory")
                    }
                    .addOnFailureListener { e ->
                        Log.e("QuizScreen", "Score save FAILED", e)

navController.navigate("results/$score/$subject/$subcategory")
                    }
            }
        }
    },
    modifier = Modifier.fillMaxWidth(),
    enabled = selectedAnswer.isNotEmpty()
```

13.Λογική αξιολόγησης απάντησης, πλοήγησης ερωτήσεων και αποθήκευσης αποτελέσματος στο Firebase.

Το κάτω μέρος της οθόνης καταλαμβάνει το κουμπί πλοήγησης ("Next Question" ή "Submit Quiz"), το οποίο ενεργοποιείται μόνο εφόσον ο χρήστης έχει επιλέξει κάποια απάντηση (`enabled = selectedAnswer.isNotEmpty()`). Όταν ο χρήστης πατά το κουμπί, η εφαρμογή ελέγχει πρώτα αν η επιλεγμένη απάντηση ταυτίζεται με τη σωστή (`currentQuestion.correctAnswer`) και, αν ναι, αυξάνει το σκορ κατά μία μονάδα. Στη συνέχεια, εάν υπάρχουν ακόμη αναπάντητες

ερωτήσεις, ο δείκτης `currentQuestionIndex` αυξάνεται κατά ένα, η επιλεγμένη απάντηση μηδενίζεται, και η οθόνη ενημερώνεται με την επόμενη ερώτηση. Αν, αντίθετα, η τρέχουσα ερώτηση είναι η τελευταία της λίστας, ενεργοποιείται η διαδικασία ολοκλήρωσης του τεστ.

Στο στάδιο ολοκλήρωσης, η εφαρμογή αναλαμβάνει να αποθηκεύσει το τελικό σκορ του χρήστη στη βάση δεδομένων. Για τον σκοπό αυτό, ανακτάται πρώτα το μοναδικό αναγνωριστικό χρήστη (UID) από το `FirebaseAuth.getInstance().currentUser`. Αν δεν υπάρχει συνδεδεμένος χρήστης (π.χ. αν το quiz έχει εκτελεστεί χωρίς προηγούμενο login), καταγράφεται σχετικό σφάλμα στα logs και η εφαρμογή πλοηγείται παρ' όλα αυτά στην οθόνη αποτελεσμάτων, ώστε να μην μπλοκάρεται η εμπειρία χρήσης. Αν όμως υπάρχει έγκυρο UID, δημιουργείται μια καταχώριση `scoreEntry` που περιλαμβάνει τη βαθμολογία και την χρονική στιγμή (`timestamp`) της προσπάθειας. Η καταχώριση αυτή αποθηκεύεται στη διαδρομή `scores/<uid>/<subject>/<subcategory>` με χρήση της `push()`, ώστε κάθε προσπάθεια να παίρνει μοναδικό κλειδί και να διατηρείται ιστορικό. Σε περίπτωση επιτυχούς αποθήκευσης, καταγράφεται ενημερωτικό μήνυμα (`Log.d`), ενώ σε αποτυχία, ένα μήνυμα σφάλματος (`Log.e`). Και στις δύο περιπτώσεις, η εφαρμογή ολοκληρώνει τη ροή πλοήγησης μεταφέροντας τον χρήστη στην οθόνη αποτελεσμάτων (`ResultsScreen`) με τα αντίστοιχα ορίσματα.

Συνολικά, η υλοποίηση της `QuizScreen` συνδυάζει δηλωτική σχεδίαση διεπαφής, δυναμική φόρτωση δεδομένων από το cloud και απλή αλλά διαφανή λογική υπολογισμού σκορ. Η χρήση του Jetpack Compose επιτρέπει την άμεση σύνδεση της κατάστασης της εφαρμογής με τη γραφική απεικόνιση, ενώ η ενσωμάτωση του Firebase Realtime Database και του Firebase Authentication εξασφαλίζει τη διατήρηση ιστορικών δεδομένων ανά χρήστη και ενότητα. Η αρχιτεκτονική αυτή καθιστά το υποσύστημα του κουίζ επεκτάσιμο, ευέλικτο και παιδαγωγικά αποτελεσματικό, προσφέροντας μια ροή αλληλεπίδρασης που είναι ταυτόχρονα απλή για τον χρήστη και τεχνικά συνεκτική για τον προγραμματιστή.

4.4 Ανάλυση της Οθόνης Αποτελεσμάτων (ResultsScreen)

Η οθόνη αποτελεσμάτων (ResultsScreen) αποτελεί το τελικό στάδιο της διαδικασίας αυτοαξιολόγησης στην εφαρμογή Edutest και είναι υπεύθυνη για την παρουσίαση της

βαθμολογίας που συγκέντρωσε ο χρήστης κατά τη διάρκεια της προσπάθειας του στο quiz. Παράλληλα, λειτουργεί ως κόμβος πλοήγησης προς δύο σημαντικές διεργασίες: την επιστροφή στην αρχική οθόνη και την προβολή του πίνακα κατάταξης (Leaderboard). Η υλοποίηση της συγκεκριμένης οθόνης χαρακτηρίζεται από απλότητα στη δομή αλλά και σαφή λειτουργικό ρόλο, ολοκληρώνοντας με ομαλό τρόπο τη συνολική εμπειρία αξιολόγησης.

```
Composable
fun ResultsScreen(
    score: Int,
    subject: String,
    subcategory: String,
    navController: NavHostController
```

14.Δήλωση του composable ResultsScreen και ορισμός παραμέτρων εισόδου (score, subject, subcategory) για την παρουσίαση αποτελεσμάτων και πλοήγηση.

Η συνάρτηση `ResultsScreen` υλοποιείται ως composable στοιχείο του Jetpack Compose και δέχεται τέσσερις παραμέτρους: το τελικό σκορ (`score`), το μάθημα (`subject`) και την υποκατηγορία (`subcategory`) που αντιστοιχούν στο quiz που ολοκληρώθηκε, καθώς και έναν `NavHostController` για την πλοήγηση στις υπόλοιπες οθόνες της εφαρμογής. Οι παράμετροι αυτές προωθούνται από την `QuizScreen` μετά την ολοκλήρωση της διαδικασίας απάντησης των ερωτήσεων, διασφαλίζοντας συνεχή ροή πληροφορίας χωρίς ανάγκη πρόσθετων ανακτήσεων από τη βάση δεδομένων.

```
Column(
    modifier = Modifier
        .fillMaxSize()
        .padding(16.dp),
    horizontalAlignment = Alignment.CenterHorizontally,
    verticalArrangement = Arrangement.spacedBy(16.dp)
) {
    Text(
        "Your score: $score",
        style = MaterialTheme.typography.headlineMedium
    )
```

15.Δομή διάταξης της οθόνης αποτελεσμάτων με χρήση Column, η οποία εμφανίζει τη συνολική βαθμολογία του χρήστη με κεντρική στοίχιση.

Η διάταξη της οθόνης βασίζεται σε ένα `Column`, το οποίο εκτείνεται σε όλη την επιφάνεια της συσκευής (`Modifier.fillMaxSize()`) και οργανώνει τα στοιχεία με ομοιόμορφη κατανομή κάθετης απόστασης μέσω της επιλογής `verticalArrangement = Arrangement.spacedBy(16.dp)`. Η στοίχιση των στοιχείων στο κέντρο (`Alignment.CenterHorizontally`) ενισχύει την καθαρότητα και την ευανάγνωστη παρουσίαση

της βαθμολογίας, προσφέροντας μια οπτικά ισορροπημένη και λειτουργικά αποτελεσματική διάταξη.

Στο επάνω μέρος της οθόνης εμφανίζεται η συνολική βαθμολογία του χρήστη μέσω ενός `Text` composable, το οποίο χρησιμοποιεί τη γραφική τυποποίηση `MaterialTheme.typography.headlineMedium`. Η επιλογή αυτή συνάδει με τις οδηγίες του Material Design για την ανάδειξη σημαντικών πληροφοριών και δίνει έμφαση στο αποτέλεσμα της προσπάθειας του χρήστη.

```
Button(
    onClick = { navController.navigate("home") },
    modifier = Modifier.fillMaxWidth()
) {
    Text("Back to Home")
}

Button(
    onClick = {
        navController.navigate("leaderboard/$subject/$subcategory")
    },
    modifier = Modifier.fillMaxWidth()
) {
    Text("View Leaderboard")
}
```

16.Κουμπιά πλοήγησης της οθόνης αποτελεσμάτων, τα οποία επιτρέπουν την επιστροφή στην αρχική οθόνη ή τη μετάβαση στον πίνακα κατάταξης.

Αμέσως μετά, η οθόνη παρέχει δύο βασικά κουμπιά ενεργειών (`Button`), τα οποία εξυπηρετούν τις πιο κρίσιμες λειτουργίες που χρειάζεται ο χρήστης μετά την ολοκλήρωση του quiz:

Επιστροφή στην Αρχική Οθόνη (Home Screen)

Η πρώτη επιλογή επιτρέπει στον χρήστη να επανέλθει στην αρχική οθόνη της εφαρμογής, όπου μπορεί να επιλέξει διαφορετικό μάθημα ή να πραγματοποιήσει άλλη προσπάθεια αξιολόγησης. Η λειτουργία αυτή υλοποιείται μέσω της εντολής: `navController.navigate("home".` Η εντολή αυτή ενεργοποιεί τον μηχανισμό πλοήγησης του Jetpack Compose και επαναφέρει τον χρήστη στη ρίζα της εφαρμογής.

Μετάβαση στον Πίνακα Κατάταξης (Leaderboard)

Το δεύτερο κουμπί κατευθύνει τον χρήστη στη σελίδα κατάταξης που αντιστοιχεί στο μάθημα και στην υποκατηγορία που μόλις ολοκληρώθηκαν. Η πλοήγηση πραγματοποιείται με την εντολή: `navController.navigate("leaderboard/$subject/$subcategory")`

Η χρήση "dynamic routing" επιτρέπει στον Leaderboard να φορτώσει αποκλειστικά τις βαθμολογίες των χρηστών για το συγκεκριμένο γνωστικό αντικείμενο. Με αυτόν τον τρόπο, ο χρήστης μπορεί να συγκρίνει άμεσα την επίδοσή του με άλλους συμμετέχοντες, ενισχύοντας την παρακίνηση και παρέχοντας μια διάσταση υγιούς ανταγωνισμού.

Η απλότητα της `ResultsScreen` δεν αποτελεί ένδειξη έλλειψης λειτουργικότητας, αλλά συνειδητή επιλογή ακολουθώντας τις αρχές του UX design. Μετά την ολοκλήρωση μιας γνωστικά απαιτητικής διαδικασίας (το quiz), ο χρήστης χρειάζεται μια καθαρή, εύληπτη και χωρίς περιττά στοιχεία οθόνη που:

- παρουσιάζει άμεσα το αποτέλεσμα,
- προσφέρει δύο σαφείς και χρήσιμες επιλογές,
- δεν αποσπά την προσοχή με δευτερεύοντα στοιχεία.

Επιπλέον, είναι σημαντικό ότι η `ResultsScreen` δεν επικοινωνεί με το Firebase· όλη η διαδικασία αποθήκευσης των αποτελεσμάτων πραγματοποιείται ήδη στο τέλος του quiz μέσα στην `QuizScreen`. Η `ResultsScreen` λειτουργεί αποκλειστικά ως οθόνη προβολής και πλοήγησης, με αποτέλεσμα να παραμένει εξαιρετικά ελαφριά και αποδοτική στη χρήση.

Συνολικά, η `ResultsScreen` συμβάλλει στην ομαλή και συνεκτική εμπειρία του χρήστη, ολοκληρώνοντας τον κύκλο της αυτοαξιολόγησης με τρόπο που είναι εργονομικός, κατανοητός και πιστός στις βέλτιστες πρακτικές σχεδίασης εκπαιδευτικών εφαρμογών.

4.5 Ανάλυση της Οθόνης Κατάταξης (LeaderboardScreen)

```
@Composable
fun LeaderboardScreen(
    subject: String,
    subcategory: String,
    navController: NavHostController? = null
) {
    val listState = rememberLazyListState()
    data class ScoreEntry(val userId: String, val score: Int, val timestamp:
Long)

    var scores by remember { mutableStateOf(listOf<ScoreEntry>()) }
    var isLoading by remember { mutableStateOf(true) }

    // Use Firebase UID for current user
    val currentUid = FirebaseAuth.getInstance().currentUser?.uid
```

17.Ορισμός της οθόνης πίνακα κατάταξης (Leaderboard) και αρχικοποίηση της κατάστασης για τη διαχείριση των βαθμολογιών των χρηστών.

Η οθόνη κατάταξης (`LeaderboardScreen`) αποτελεί κρίσιμο στοιχείο της εφαρμογής, καθώς επιτρέπει τη σύγκριση επιδόσεων μεταξύ χρηστών και ενισχύει το στοιχείο της παρακίνησης στη μαθησιακή διαδικασία. Η υλοποίησή της βασίζεται στο Jetpack Compose και αξιοποιεί πλήρως το Firebase Realtime Database καθώς και τον μηχανισμό ταυτοποίησης Firebase Authentication για την ανάδειξη του τρέχοντος χρήστη. Η συνάρτηση ορίζεται ως @Composable και δέχεται τις παραμέτρους *subject*, *subcategory* και προαιρετικά έναν *NavHostController* για πλοήγηση. Στον πυρήνα της χρησιμοποιεί μια τοπική δομή δεδομένων ScoreEntry, η οποία περιλαμβάνει πληροφορίες για τον χρήστη (`userId`), τη βαθμολογία και τη χρονική στιγμή καταγραφής της επίδοσης (`timestamp`). Η οθόνη διατηρεί εσωτερική κατάσταση μέσω των `remember` και `mutableStateOf`, ώστε να αποθηκεύονται δυναμικά τόσο οι βαθμολογίες όσο και η κατάσταση φόρτωσης.

```
aunchedEffect(subject, subcategory) {
    val scoresRef = FirebaseDatabase.getInstance().getReference("scores")
    scoresRef.get()
        .addOnSuccessListener { snapshot ->
            val list = mutableListOf<ScoreEntry>()
            snapshot.children.forEach { userSnapshot ->
                val userId = userSnapshot.key ?: return@forEach
                val subScores =
userSnapshot.child(subject).child(subcategory)

                // Highest score for this user on this subcategory
                val maxScoreSnap = subScores.children.maxByOrNull {
                    (it.child("score").value as? Long) ?: 0L
                }

                val scoreValue = (maxScoreSnap?.child("score")?.value as?
Long)?.toInt() ?: 0
                val timestamp = (maxScoreSnap?.child("timestamp")?.value as?
Long)
                    ?: 0L

                // Include only users who have at least one attempt
(timestamp > 0)
                if (timestamp > 0L) {
                    list.add(ScoreEntry(userId, scoreValue, timestamp))
                }
            }

            scores = list.sortedWith(
                compareByDescending<ScoreEntry> { it.score }
                    .thenByDescending { it.timestamp }
            )
            isLoading = false
        }
        .addOnFailureListener { e ->
            isLoading = false
            Log.e("LeaderboardScreen", "Failed to load scores", e)
        }
}
```

18.Ανάκτηση και επεξεργασία των βαθμολογιών των χρηστών από το Firebase Realtime Database για τη δημιουργία του πίνακα κατάταξης.

Η ανάκτηση των δεδομένων πραγματοποιείται μέσα σε ένα `LaunchedEffect`, το οποίο ενεργοποιείται όταν μεταβάλλονται τα επιλεγμένα *subject* ή *subcategory*. Η εφαρμογή διατρέχει τον κόμβο `scores` της βάσης δεδομένων και για κάθε χρήστη εντοπίζει όλες τις προηγούμενες προσπάθειές του στο συγκεκριμένο μάθημα και υποκατηγορία. Από αυτές τις προσπάθειες εξάγεται η καλύτερη (μέγιστη) επίδοση, συγκρίνοντας τις τιμές του πεδίου `score`. Η σχετική λειτουργία υλοποιείται με τη μέθοδο `maxByOrNull`, ενώ τα δεδομένα μετατρέπονται σε αντικείμενα `ScoreEntry`. Η λίστα που σχηματίζεται ταξινομείται με φθίνουσα σειρά βάσει της βαθμολογίας και, σε περίπτωση ισοβαθμίας, με φθίνουσα σειρά χρονικής σήμανσης. Με αυτόν

τον τρόπο εξασφαλίζεται ότι οι πιο πρόσφατες υψηλές επιδόσεις εμφανίζονται υψηλότερα στον πίνακα. Παράλληλα, η εφαρμογή εντοπίζει το μοναδικό αναγνωριστικό (UID) του τρέχοντος χρήστη μέσω του Firebase Authentication, επιτρέποντας την ανάδειξη της προσωπικής του θέσης στο σύνολο της κατάταξης.

```
LazyColumn(
    state = listState,
    modifier = Modifier
        .fillMaxSize()
        .padding(16.dp),
    verticalArrangement = Arrangement.spacedBy(8.dp)
) {
    item {
        navController?.let {
            Button(onClick = { it.navigate("home") }, modifier =
Modifier.fillMaxWidth()) {
                Text("Back to Home")
            }
        }
    }

    item {
        Text(
            "Leaderboard: $subject → $subcategory",
            style = MaterialTheme.typography.headlineMedium
        )
    }

    // Top scores (scrollable)
    itemsIndexed(displayScores) { index, entry ->
        val isCurrentUser = entry.userId == currentUid
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .background(if (isCurrentUser) Color(0xFFE0F7FA) else
Color.Transparent)
                .padding(8.dp),
            horizontalArrangement = Arrangement.SpaceBetween
        ) {
            Column {
                // Rank shown here is index+1 for the visible top-10 list;
                // if this is the current user and they're actually outside
top-10,
                // we still show their real rank in the bottom "Your Score"
card.
                val visibleRank = index + 1
                Text(
                    text = "$visibleRank. User: ${entry.userId} - Score:
${entry.score}",
                    style = MaterialTheme.typography.bodyLarge
                )
                Text(
                    text = "Time:
${dateFormat.format(Date(entry.timestamp))}",
                    style = MaterialTheme.typography.bodySmall
                )
            }
        }
    }
```

19.Δυναμική δημιουργία και παρουσίαση του πίνακα κατάταξης με χρήση LazyColumn και οπτική επισήμανση του τρέχοντος χρήστη.

Η διεπαφή χρήστη βασίζεται σε ένα `LazyColumn`, το οποίο προσφέρει αποδοτική κύλιση ακόμη και για μεγάλους όγκους δεδομένων. Στην κορυφή τοποθετείται ένα κουμπί επιστροφής στην αρχική οθόνη, ενώ ακολουθεί ο τίτλος της κατάταξης. Στη συνέχεια αποδίδονται οι δέκα καλύτερες επιδόσεις, οι οποίες εμφανίζονται ως ξεχωριστά στοιχεία λίστας. Κάθε στοιχείο περιλαμβάνει πληροφορίες για τη βαθμολογία, την κατάταξη, το UID του χρήστη και τη χρονική στιγμή καταγραφής, η οποία μορφοποιείται μέσω του `SimpleDateFormat`. Για τον τρέχοντα χρήστη εφαρμόζεται ειδικός χρωματισμός φόντου, γεγονός που διευκολύνει την οπτική αναγνώριση της θέσης του.

```
if (showBottomYourScore && currentUserScore != null && currentUserRank > 0)
{
    item {
        HorizontalDivider(
            color = Color.Gray,
            thickness = 2.dp,
            modifier = Modifier.padding(vertical = 4.dp)
        )
    }
    item {
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .background(Color(0xFFE0F7FA))
                .padding(8.dp),
            horizontalArrangement = Arrangement.SpaceBetween
        ) {
            Column {
                Text(
                    text = "$currentUserRank. Your Score:
${currentUserScore.score}",
                    style = MaterialTheme.typography.bodyLarge
                )
                Text(
                    text = "Time:
${dateFormat.format(Date(currentUserScore.timestamp))}",
                    style = MaterialTheme.typography.bodySmall
                )
            }
        }
    }
```

20.Προβολή της προσωπικής επίδοσης του χρήστη εκτός της πρώτης δεκάδας, με οπτική διαχωριστική γραμμή και έμφαση στη θέση κατάταξης.

Ιδιαίτερη μέριμνα δίνεται στην περίπτωση όπου ο τρέχων χρήστης δεν περιλαμβάνεται στις πρώτες δέκα θέσεις. Στην περίπτωση αυτή υπολογίζεται η συνολική του κατάταξη και εμφανίζεται ειδικό τμήμα στο κάτω μέρος της λίστας με την προσωπική του επίδοση. Για να διασφαλιστεί η ευκολία εντοπισμού αυτής της πληροφορίας, χρησιμοποιείται η λειτουργία `scrollToItem` του `LazyListState`, η οποία μετακινεί αυτόματα τη λίστα προς το τμήμα όπου εμφανίζεται η επίδοση του χρήστη. Έτσι, η εφαρμογή εξασφαλίζει άμεση και φιλική προς τον χρήστη πρόσβαση στις προσωπικές του πληροφορίες, ακόμη και σε μεγάλα σύνολα δεδομένων.

Συνολικά, η υλοποίηση της `LeaderboardScreen` συνιστά έναν ολοκληρωμένο μηχανισμό διαχείρισης και προβολής επιδόσεων, ο οποίος αξιοποιεί προηγμένες τεχνικές ανάκτησης και ταξινόμησης δεδομένων από το Firebase, χρησιμοποιεί βελτιστοποιημένα εργαλεία διεπαφής του Jetpack Compose και επιτυγχάνει υψηλό βαθμό διαδραστικότητας και εξατομίκευσης της εμπειρίας χρήστη. Η προσέγγιση αυτή καθιστά την οθόνη κατάταξης όχι μόνο λειτουργική αλλά και παιδαγωγικά αποτελεσματική, ενισχύοντας τη συμμετοχικότητα και τον υγιή ανταγωνισμό μεταξύ των χρηστών της εφαρμογής Edutest.

4.6 Οθόνη Σύνδεσης Διαχειριστή (LoginScreen)

```
@Composable
fun LoginScreen(navController: NavHostController) {
    var email by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var error by remember { mutableStateOf<String?>(null) }
    var loading by remember { mutableStateOf(false) }

@Composable
fun LoginScreen(navController: NavHostController) {
    var email by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var error by remember { mutableStateOf<String?>(null) }
    var loading by remember { mutableStateOf(false) }

    fun signIn() {
        loading = true
        error = null
        FirebaseAuth.getInstance()
            .signInWithEmailAndPassword(email.trim(), password)
            .addOnSuccessListener {
                loading = false
                navController.navigate("home") {
                    popUpTo("home") { inclusive = true }
                }
            }
            .addOnFailureListener { e ->
                loading = false
                error = e.message ?: "Login failed"
                Log.e("LoginScreen", "signIn failed", e)
            }
    }

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.spacedBy(12.dp)
    ) {
        Text("Admin Login", style = MaterialTheme.typography.headlineMedium)

        OutlinedTextField(
            value = email,
            onValueChange = { email = it },
            label = { Text("Email") },
            modifier = Modifier.fillMaxWidth()
        )
        OutlinedTextField(
            value = password,
            onValueChange = { password = it },
            label = { Text("Password") },
            visualTransformation = PasswordVisualTransformation(),
            modifier = Modifier.fillMaxWidth()
        )

        if (error != null) {
            Text(error!!, color = MaterialTheme.colorScheme.error)
```

```
        }

        Button(
            onClick = { signIn() },
            enabled = !loading && email.isNotBlank() &&
password.isNotBlank(),
            modifier = Modifier.fillMaxWidth()
        ) {
            Text(if (loading) "Signing in..." else "Sign In")
        }

        TextButton(
            onClick = { navController.navigate("home") },
            modifier = Modifier.fillMaxWidth()
        ) {
            Text("Back to Home")
        }
    }
}
```

***21**.Υλοποίηση της οθόνης σύνδεσης διαχειριστή (Admin Login), με διαχείριση κατάστασης εισόδου, έλεγχο εγκυρότητας στοιχείων και αυθεντικοποίηση μέσω Firebase Authentication.*

Η οθόνη `LoginScreen` αποτελεί το σημείο ελεγχόμενης πρόσβασης στην περιοχή διαχείρισης της εφαρμογής, προσφέροντας δυνατότητα εισόδου μόνο σε εξουσιοδοτημένους χρήστες μέσω των μηχανισμών αυθεντικοποίησης του Firebase Authentication. Η χρήση ξεχωριστής οθόνης σύνδεσης ενισχύει την ασφάλεια της εφαρμογής και διασφαλίζει ότι μόνο οι διαχειριστές μπορούν να μεταβάλλουν δεδομένα, όπως την εισαγωγή ή επεξεργασία ερωτήσεων.

Ο σχεδιασμός βασίζεται εξ ολοκλήρου στο Jetpack Compose και υιοθετεί πλήρως δηλωτικές αρχές ανάπτυξης. Στην αρχή του composable δημιουργούνται μέσω `remember` οι καταστάσεις (`email`, `password`, `error`, `loading`) που διατηρούνται και ενημερώνονται δυναμικά καθώς ο χρήστης αλληλεπιδρά με τα πεδία εισόδου. Η μεταβλητή `loading` ελέγχει εάν βρίσκεται σε εξέλιξη διαδικασία σύνδεσης προς το Firebase, ενώ η μεταβλητή `error` χρησιμοποιείται για την απεικόνιση σφαλμάτων που ενδεχομένως προκύψουν.

Στο εσωτερικό του composable ορίζεται η βοηθητική συνάρτηση `signIn()`, η οποία υλοποιεί τη διαδικασία αυθεντικοποίησης. Η μέθοδος `signInWithEmailAndPassword` της κλάσης `FirebaseAuth` καλείται με τα στοιχεία που έχει εισαγάγει ο χρήστης (email και password). Λόγω του ασύγχρονου χαρακτήρα της, η μέθοδος επιστρέφει listeners:

- Με την `addOnSuccessListener`, η μεταβλητή `loading` μηδενίζεται και η εφαρμογή μεταβαίνει στην αρχική οθόνη (`home`), χρησιμοποιώντας το

`navController.navigate("home")` σε συνδυασμό με `popUpTo("home") { inclusive = true }`, ώστε να αποτρέπεται η επιστροφή στην οθόνη σύνδεσης με το κουμπί “Back”.

- Με την `addOnFailureListener`, εμφανίζεται μήνυμα σφάλματος μέσω της μεταβλητής `error`, και καταγράφεται λεπτομερές μήνυμα στο logcat με την αιτία της αποτυχίας. Η πρακτική αυτή διευκολύνει τη διάγνωση προβλημάτων κατά την ανάπτυξη.

Η διεπαφή χρήστη έχει σχεδιαστεί με απλότητα και λειτουργικότητα. Περιλαμβάνει δύο `OutlinedTextField` για την εισαγωγή email και κωδικού πρόσβασης, με το δεύτερο να χρησιμοποιεί `PasswordVisualTransformation` για απόκρυψη χαρακτήρων. Κάτω από τα πεδία εμφανίζεται δυναμικά μήνυμα σφάλματος, όταν η διαδικασία σύνδεσης αποτύχει. Το κουμπί “Sign In” ενεργοποιείται μόνο όταν υπάρχουν έγκυρες τιμές στα πεδία και δεν βρίσκεται σε εξέλιξη άλλη προσπάθεια σύνδεσης. Ο χρήστης λαμβάνει επιπρόσθετα οπτική ανατροφοδότηση μέσω του κειμένου του κουμπιού (“Signing in…”), το οποίο εμφανίζεται όσο η μεταβλητή `loading` είναι ενεργή.

Στο κάτω μέρος της οθόνης υπάρχει ένα `TextButton` “Back to Home”, το οποίο επιτρέπει στον χρήστη να επιστρέψει στην αρχική οθόνη χωρίς να ολοκληρώσει τη διαδικασία σύνδεσης. Η δυνατότητα αυτή ενισχύει τη χρηστικότητα για την περίπτωση όπου ο χρήστης δεν επιθυμεί τελικά να αποκτήσει πρόσβαση ως διαχειριστής.

Συνολικά, η `LoginScreen` επιτελεί έναν κρίσιμο ρόλο στη δομή της εφαρμογής Edutest: αποτελεί το κέντρο ελέγχου πρόσβασης για διαχειριστικές λειτουργίες, προσφέροντας μια ασφαλή και απλή διαδικασία σύνδεσης, παρέχοντας οπτική ανατροφοδότηση στον χρήστη, και διασφαλίζοντας ότι μόνο εξουσιοδοτημένα άτομα μπορούν να τροποποιούν το εκπαιδευτικό περιεχόμενο, τις ερωτήσεις και τα δεδομένα της εφαρμογής.

4.7 Πίνακας Διαχειριστή (AdminDashboardScreen)

Ο πίνακας διαχειριστή (`AdminDashboardScreen`) αποτελεί το κεντρικό εργαλείο συντήρησης και επέκτασης του εκπαιδευτικού περιεχομένου της εφαρμογής Edutest. Μέσω αυτής της οθόνης, ο διαχειριστής έχει τη δυνατότητα να προσθέτει, να τροποποιεί ή να διαγράφει μαθήματα (subjects), υποκατηγορίες (subcategories) και ερωτήσεις, καθώς και να εισάγει μαζικά δεδομένα από αρχεία JSON. Η οθόνη αυτή συνδυάζει σύνθετη λογική διαχείρισης δεδομένων με μια όσο το δυνατόν απλή και λειτουργική διεπαφή, ώστε να μπορεί να αξιοποιηθεί από χρήστες με βασικές γνώσεις χειρισμού υπολογιστών.

```
@Composable
fun AdminDashboardScreen(navController: NavHostController) {
    val isAdmin = rememberIsAdmin()
    if (!isAdmin) {
        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(16.dp),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.spacedBy(12.dp)
        ) {
            Text("You must be an admin to access this screen.", style =
MaterialTheme.typography.titleMedium)
            Button(onClick = { navController.navigate("login") }, modifier =
Modifier.fillMaxWidth()) {
                Text("Go to Admin Login")
            }
            Button(onClick = { navController.navigate("home") }, modifier =
Modifier.fillMaxWidth()) {
                Text("Back to Home")
            }
        }
        return
    }
```

22.Έλεγχος δικαιωμάτων πρόσβασης διαχειριστή (admin gate) πριν την εμφάνιση του πίνακα διαχείρισης, με ανακατεύθυνση μη εξουσιοδοτημένων χρηστών.

Στο ανώτερο επίπεδο ορίζεται το composable `AdminDashboardScreen(navController: NavHostController)`, το οποίο λειτουργεί ως "φύλακας" πρόσβασης στον πίνακα διαχειριστή. Η συνάρτηση καλεί το βοηθητικό `rememberIsAdmin()`, το οποίο ελέγχει, μέσω Firebase Realtime Database, αν ο τρέχων χρήστης (βάσει του UID του από το Firebase Authentication) έχει καταχωριστεί στον κόμβο `admins`. Εάν ο χρήστης δεν είναι διαχειριστής, προβάλλεται μια απλή ενημερωτική οθόνη με μήνυμα ότι απαιτούνται δικαιώματα admin, καθώς και δύο κουμπιά για

μετάβαση στην οθόνη σύνδεσης (`login`) ή επιστροφή στην αρχική οθόνη (`home`). Μόνο όταν η συνθήκη `isAdmin` είναι αληθής, καλείται η συνάρτηση `AdminDashboardContent`, η οποία περιέχει όλη τη λειτουργικότητα του πίνακα διαχείρισης. Με τον τρόπο αυτό διασφαλίζεται ότι η πρόσβαση σε κρίσιμες λειτουργίες αλλαγής περιεχομένου είναι αυστηρά ελεγχόμενη.

```
val isPreview = LocalInspectionMode.current

val context = LocalContext.current
val snackbarHostState = remember { SnackbarHostState() }
val scope = rememberCoroutineScope()

var subjects by remember { mutableStateOf(listOf<String>()) }
var selectedSubject by remember { mutableStateOf<String?>(null) }
var subcategories by remember { mutableStateOf(listOf<String>()) }
var selectedSubcategory by remember { mutableStateOf<String?>(null) }

var newSubjectName by remember { mutableStateOf("") }
var newSubcategoryName by remember { mutableStateOf("") }

var questions by remember { mutableStateOf(listOf<Question>()) }
var editingQuestionIndex by remember { mutableStateOf<Int?>(null) }
var newQuestionText by remember { mutableStateOf("") }
var newQuestionOptions by remember { mutableStateOf("") }

val database = remember { FirebaseDatabase.getInstance() }
val subjectsRef: DatabaseReference = remember {
database.getReference("subjects") }
val questionsRef: DatabaseReference = remember {
database.getReference("questions") }
```

23.Δήλωση και διαχείριση της κατάστασης (state) της οθόνης διαχείρισης, συμπεριλαμβανομένων των μεταβλητών περιεχομένου, των αναφορών Firebase και των βοηθητικών μηχανισμών διεπαφής.

Η κύρια λειτουργικότητα υλοποιείται στο composable `AdminDashboardContent(navController: NavHostController)`. Αρχικά ορίζονται οι βασικές καταστάσεις (state) της οθόνης: λίστα μαθημάτων (`subjects`), επιλεγμένο μάθημα (`selectedSubject`), λίστα υποκατηγοριών (`subcategories`), επιλεγμένη υποκατηγορία (`selectedSubcategory`), καθώς και μεταβλητές για τη διαχείριση των πεδίων εισαγωγής νέου μαθήματος και υποκατηγορίας (`newSubjectName`, `newSubcategoryName`). Σε επίπεδο ερωτήσεων χρησιμοποιείται λίστα τύπου `List<Question>` για τα διαθέσιμα quiz ανά υποκατηγορία, καθώς και μεταβλητές `editingQuestionIndex`, `newQuestionText` και `newQuestionOptions` για την επεξεργασία ή εισαγωγή νέων ερωτήσεων. Για τις επικοινωνίες με το Firebase χρησιμοποιούνται δύο βασικές αναφορές (`DatabaseReference`): μία στον κόμβο `subjects` και μία στον κόμβο `questions`, ώστε να διαχωρίζεται λογικά η λίστα των διαθέσιμων μαθημάτων από τη δομή των

ερωτήσεων. Παράλληλα, μέσω `SnackbarHostState` και `rememberCoroutineScope` υλοποιείται μηχανισμός εμφάνισης ενημερωτικών μηνυμάτων (snackbars) προς τον διαχειριστή, π.χ. σε επιτυχή εισαγωγή δεδομένων ή σφάλμα ανάγνωσης αρχείου.

```
fun reloadSubjectsAndCurrentBranch() {
    if (isPreview) return
    subjectsRef.get()
        .addOnSuccessListener { snap ->
            subjects = snap.children.mapNotNull { it.value?.toString() }

            selectedSubject?.let { subj ->
                questionsRef.child(subj).get()
                    .addOnSuccessListener { subSnap ->
                        subcategories = subSnap.children.mapNotNull { it.key
}

                        selectedSubcategory?.let { sub ->
                            questionsRef.child(subj).child(sub).get()
                                .addOnSuccessListener { qSnap ->
                                    val list = qSnap.children.mapNotNull {
child ->
                                        val text =
child.child("questionText").getValue(String::class.java)
                                        val opts =
child.child("options").getValue() as? List<String>
                                        val correct =
child.child("correctAnswer").getValue(String::class.java)
                                        if (text != null && opts != null &&
correct != null) Question(text, opts, correct) else null
                                    }
                                    questions = list
                                }
                        }
                    }
            }
        }
```

24.Βοηθητική συνάρτηση επαναφόρτωσης των θεμάτων, υποκατηγοριών και ερωτήσεων από τη βάση δεδομένων Firebase, με συγχρονισμό της κατάστασης της διεπαφής.

Ένα σημαντικό στοιχείο της υλοποίησης είναι η βοηθητική συνάρτηση `reloadSubjectsAndCurrentBranch()`. Η συνάρτηση αυτή επαναφορτώνει από το Firebase τη λίστα των μαθημάτων και, εφόσον υπάρχει επιλεγμένο μάθημα και υποκατηγορία, ανανεώνει αντίστοιχα τις υποκατηγορίες και τις ερωτήσεις που εμφανίζονται στην οθόνη. Η προσέγγιση αυτή διασφαλίζει ότι, μετά από ενέργειες όπως εισαγωγή JSON, μετονομασία ή διαγραφή, η προβολή παραμένει συγχρονισμένη με την πραγματική κατάσταση της βάσης δεδομένων. Σε περίπτωση αποτυχίας, εμφανίζεται σχετικό μήνυμα μέσω snackbar, προσφέροντας άμεση ανατροφοδότηση στον διαχειριστή.

```
val launcher = rememberLauncherForActivityResult(
    contract = ActivityResultContracts.GetContent()
) { uri ->
    uri?.let {
        try {
            context.contentResolver.openInputStream(uri)?.use { inputStream
->
                val reader = InputStreamReader(inputStream)
                val gson = Gson()
                val quizData = gson.fromJson<QuizData>(reader, object :
TypeToken<QuizData>() {}.type)

                // Append to Firebase (push), won't erase existing content
                uploadQuizDataToFirebase(quizData, subjectsRef,
questionsRef)

                scope.launch {
                    snackbarHostState.showSnackbar("☑ Successfully
imported quiz data!")
                }
                reloadSubjectsAndCurrentBranch()
            }
        } catch (e: Exception) {
            Log.e("AdminDashboard", "Error reading JSON file", e)
            scope.launch {
                snackbarHostState.showSnackbar("❌ Failed to import JSON
file.")
            }
        }
    }
}
```

25,Μηχανισμός επιλογής και εισαγωγής αρχείου JSON για μαζική προσθήκη δεδομένων στη βάση Firebase, με αυτόματη ενημέρωση της διεπαφής διαχειριστή.

Ιδιαίτερα κρίσιμο σημείο για την ευχρηστία της εφαρμογής είναι ο μηχανισμός εισαγωγής δεδομένων από αρχείο JSON. Η λειτουργία αυτή υλοποιείται με τη χρήση `rememberLauncherForActivityResult` και του `ActivityResultContracts.GetContent`, που προβάλλει picker αρχείων στο Android και επιτρέπει την επιλογή ενός JSON από το σύστημα αρχείων της συσκευής ή του emulator. Μόλις ο χρήστης επιλέξει αρχείο, το content resolver ανοίγει stream, το οποίο διαβάζεται μέσω `InputStreamReader` και αναλύεται με τη βιβλιοθήκη `Gson` σε αντικείμενο τύπου `QuizData`. Η δομή `QuizData` περιλαμβάνει μία λίστα `SubjectImport`, όπου κάθε αντικείμενο φέρει όνομα μαθήματος και τις αντίστοιχες υποκατηγορίες με τις ερωτήσεις τους. Η συνάρτηση `uploadQuizDataToFirebase` αναλαμβάνει την αποθήκευση

αυτών των δεδομένων στο Firebase, με χρήση `push()` για προσθήκη (append) χωρίς διαγραφή υπάρχοντος περιεχομένου. Κατόπιν επιτυχούς εισαγωγής, εμφανίζεται μήνυμα επιβεβαίωσης ("Successfully imported quiz data") και καλείται `reloadSubjectsAndCurrentBranch()` για να ανανεωθεί η προβολή. Σε περίπτωση εξαίρεσης (π.χ. κακόμορφο JSON) καταγράφεται σφάλμα στο logcat και εμφανίζεται αντίστοιχο ενημερωτικό snackbar προς τον χρήστη.

```
LaunchedEffect(Unit) {
    if (isPreview) {
        subjects = listOf("Math", "HTML")
    } else {
        subjectsRef.get().addOnSuccessListener { snapshot ->
            subjects = snapshot.children.mapNotNull { it.value?.toString() }
        }.addOnFailureListener { e ->
            Log.e("FirebaseAdmin", "Error loading subjects", e)
        }
    }
}

// When subject changes, load its subcategories; reset subcategory/questions
LaunchedEffect(selectedSubject) {
    selectedSubject?.let { subject ->
        if (isPreview) {
            subcategories = listOf("Algebra", "Geometry")
        } else {
            questionsRef.child(subject).get().addOnSuccessListener {
snapshot ->
                subcategories = snapshot.children.mapNotNull { it.key }
            }
        }
        selectedSubcategory = null
        questions = emptyList()
    }
}

// When subcategory changes, load its questions
LaunchedEffect(selectedSubject, selectedSubcategory) {
    if (selectedSubject != null && selectedSubcategory != null) {
        if (isPreview) {
            questions = listOf(
                Question("Preview Q1", listOf("A", "B", "C", "D"), "A"),
                Question("Preview Q2", listOf("1", "2", "3", "4"), "2")
            )
        } else {
            val subRef =
questionsRef.child(selectedSubject!!).child(selectedSubcategory!!)
            subRef.get().addOnSuccessListener { snapshot ->
                val list = snapshot.children.mapNotNull { child ->
                    val text =
child.child("questionText").getValue(String::class.java)
                    val opts = child.child("options").getValue() as?
List<String>
                    val correct =
child.child("correctAnswer").getValue(String::class.java)
                    if (text != null && opts != null && correct != null)
Question(text, opts, correct) else null
                }
                questions = list
            }
        }
    }
}
```

26.Υλοποίηση μηχανισμού επαναφόρτωσης δεδομένων βάσει αλλαγών κατάστασης (state-driven updates), αξιοποιώντας το LaunchedEffect του Jetpack Compose για την ανάκτηση περιεχομένου από το Firebase Realtime Database.

Η αρχική φόρτωση των μαθημάτων πραγματοποιείται μέσα από ένα `LaunchedEffect(Unit)`, το οποίο κατά την εκκίνηση της οθόνης ανακτά από τον κόμβο `subjects` όλα τα αποθηκευμένα μαθήματα και τα αποδίδει στη λίστα `subjects`. Επιπρόσθετα, δύο ακόμη `LaunchedEffect` μπλοκ φροντίζουν για τη δυναμική φόρτωση των υποκατηγοριών και των ερωτήσεων: όταν αλλάζει το `selectedSubject`, ανακτώνται τα παιδιά του αντίστοιχου κλάδου στον κόμβο `questions`, τα οποία αντιστοιχούν σε υποκατηγορίες. Όταν οριστεί και `selectedSubcategory`, η εφαρμογή διαβάζει τις ερωτήσεις αυτού του κλάδου, δημιουργώντας αντικείμενα `Question` από τα πεδία `questionText`, `options` και `correctAnswer`. Η λογική αυτή εξασφαλίζει ότι ο πίνακας διαχειριστή παραμένει συνεπής με τη δομή της βάσης, ανεξάρτητα από το πόσο έχει επεκταθεί το περιεχόμενο.

```
Scaffold(
    snackbarHost = { SnackbarHost(snackbarHostState) }
) { innerPadding ->
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(innerPadding)
            .padding(16.dp),
        verticalArrangement = Arrangement.spacedBy(16.dp)
    ) {
        // Top actions
        Button(
            onClick = { launcher.launch("application/json") },
            modifier = Modifier.fillMaxWidth()
        ) { Text("Import JSON File") }

        Button(
            onClick = { navController.navigate("home") },
            modifier = Modifier.fillMaxWidth()
        ) { Text("Back to Home") }

        Text("Admin Dashboard", style =
MaterialTheme.typography.headlineMedium)

        HorizontalDivider()

        // ===== Scrollable content =====
        LazyColumn(
            modifier = Modifier.fillMaxSize(),
            verticalArrangement = Arrangement.spacedBy(16.dp)
        )
```

27.Δομή οθόνης διαχείρισης (Admin Dashboard) με χρήση Scaffold, ενσωματωμένο μηχανισμό ειδοποιήσεων (Snackbar) και κατακόρυφη διάταξη ενεργειών και περιεχομένου.

Το οπτικό τμήμα της οθόνης υλοποιείται μέσω ενός `Scaffold`, στο οποίο έχει δηλωθεί `SnackbarHost` για την εμφάνιση ενημερωτικών μηνυμάτων. Στην κορυφή της διάταξης (μέσα σε `Column`) εμφανίζονται δύο βασικά κουμπιά: “Import JSON File” για την εκκίνηση της διαδικασίας εισαγωγής αρχείου, και “Back to Home” για επιστροφή στην αρχική οθόνη της εφαρμογής. Ακολουθεί τίτλος “Admin Dashboard” και ένας `Divider`, που οπτικά χωρίζει τη ζώνη ενεργειών από το κυρίως περιεχόμενο. Το υπόλοιπο της οθόνης είναι ένα `LazyColumn`, που περιλαμβάνει διαδοχικά τμήματα για τη διαχείριση μαθημάτων, υποκατηγοριών και ερωτήσεων.

```
items(subjects) { subject ->
    Card(
        modifier = Modifier
            .fillMaxWidth()
            .clickable { selectedSubject = subject }
            .background(
                if (subject == selectedSubject)
MaterialTheme.colorScheme.primary.copy(alpha = 0.1f)
                else MaterialTheme.colorScheme.surface
            ),
        elevation = CardDefaults.cardElevation(4.dp)
    ) {
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(8.dp),
            horizontalArrangement = Arrangement.SpaceBetween,
            verticalAlignment = Alignment.CenterVertically
        ) {
            Text(subject, style = MaterialTheme.typography.titleMedium)
            Row {
                IconButton(onClick = {
                    newSubjectName = subject
                    selectedSubject = subject
                }) { Icon(Icons.Filled.Edit, contentDescription = "Edit
Subject") }

                IconButton(onClick = {
                    if (!isPreview) {
                        subjectsRef.child(subject).removeValue()
                        questionsRef.child(subject).removeValue()
                    }
                    subjects = subjects.filter { it != subject }
                    if (selectedSubject == subject) {
                        selectedSubject = null
                        subcategories = emptyList()
                        selectedSubcategory = null
                        questions = emptyList()
                    }
                }) { Icon(Icons.Filled.Delete, contentDescription = "Delete
Subject") }
            }
        }
    }
}

// Add / Update Subject
item {
    Row(horizontalArrangement = Arrangement.spacedBy(8.dp)) {
        OutlinedTextField(
            value = newSubjectName,
            onValueChange = { newSubjectName = it },
            label = { Text("Subject Name") },
            modifier = Modifier.weight(1f)
        )
        Button(onClick = {
            val name = newSubjectName.trim()
```

```
            if (name.isNotEmpty()) {
                if (selectedSubject != null) {
                    // Rename: copy questions subtree to new subject, update
subjects list
                    if (!isPreview) {
                        val old = selectedSubject!!
                        questionsRef.child(old).get().addOnSuccessListener {
snap ->
                            questionsRef.child(name).setValue(snap.value)
                            subjectsRef.child(old).removeValue()
                            subjectsRef.push().setValue(name)
                            questionsRef.child(old).removeValue()
                            reloadSubjectsAndCurrentBranch()
                        }
                    }
                    subjects = subjects.map { if (it == selectedSubject)
name else it }
                    selectedSubject = null
                } else {
                    if (!isPreview) subjectsRef.push().setValue(name)
                    subjects = subjects + name
                }
                newSubjectName = ""
            }
        }) {
            Text(if (selectedSubject != null) "Update Subject" else "Add
Subject")
        }
    }
}
```

28.Διαχείριση θεματικών ενοτήτων (Subjects) στο περιβάλλον διαχειριστή, με δυνατότητες επιλογής, επεξεργασίας, διαγραφής και προσθήκης νέων θεμάτων.

Στο τμήμα των μαθημάτων, κάθε subject εμφανίζεται ως `Card` με δυνατότητα επιλογής (μέσω `clickable`) και με δύο `IconButton` για επεξεργασία (edit) και διαγραφή (delete). Η επιλογή κάρτας θέτει το `selectedSubject` και πυροδοτεί τη φόρτωση των αντίστοιχων υποκατηγοριών. Η επεξεργασία αντιγράφει το όνομα του subject στο πεδίο `newSubjectName`, ενώ η διαγραφή αφαιρεί τόσο την εγγραφή από τον κόμβο `subjects` όσο και το αντίστοιχο subtree στον κόμβο `questions`. Ακριβώς κάτω από τη λίστα μαθημάτων υπάρχει ενότητα “Add / Update Subject”, η οποία περιλαμβάνει `OutlinedTextField` για εισαγωγή ονόματος και ένα `Button` που είτε προσθέτει νέο μάθημα (με `push().setValue(name)`), είτε μετονομάζει υπάρχον subject αντιγράφοντας το υποδένδρο των ερωτήσεων στο νέο όνομα και διαγράφοντας το παλαιό. Η λογική αυτή επιτρέπει στον διαχειριστή να προσαρμόζει τη δομή των μαθημάτων χωρίς να χάνει τα ήδη καταχωρισμένα quiz.

```
if (selectedSubject != null) {
    items(subcategories) { sub ->
        Card(
            modifier = Modifier
                .fillMaxWidth()
                .clickable { selectedSubcategory = sub }
                .background(
                    if (sub == selectedSubcategory)
MaterialTheme.colorScheme.secondary.copy(alpha = 0.1f)
                    else MaterialTheme.colorScheme.surface
                ),
            elevation = CardDefaults.cardElevation(2.dp)
        ) {
            Row(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(8.dp),
                horizontalArrangement = Arrangement.SpaceBetween,
                verticalAlignment = Alignment.CenterVertically
            ) {
                Text(sub, style = MaterialTheme.typography.titleMedium)
                Row {
                    IconButton(onClick = {
                        newSubcategoryName = sub
                        selectedSubcategory = sub
                    }) { Icon(Icons.Filled.Edit, contentDescription = "Edit
Subcategory") }

                    IconButton(onClick = {
                        if (!isPreview) {

questionsRef.child(selectedSubject!!).child(sub).removeValue()
                        }
                        subcategories = subcategories.filter { it != sub }
                        if (selectedSubcategory == sub) {
                            selectedSubcategory = null
                            questions = emptyList()
                        }
                    }) { Icon(Icons.Filled.Delete, contentDescription =
"Delete Subcategory") }
                }
            }
        }
    }
```

29.Διαχείριση υποκατηγοριών ανά θεματική ενότητα, με δυνατότητες επιλογής, επεξεργασίας και διαγραφής μέσω του περιβάλλοντος διαχειριστή.

Όταν έχει επιλεγεί μάθημα, εμφανίζεται αντίστοιχα η λίστα υποκατηγοριών. Κάθε υποκατηγορία προβάλλεται επίσης ως `Card` με όνομα, καθώς και κουμπιά για επεξεργασία και διαγραφή. Η διαγραφή αφαιρεί τον συγκεκριμένο κλάδο ερωτήσεων από τον κόμβο `questions`, ενώ η επεξεργασία φορτώνει το όνομα στο `newSubcategoryName`. Ακολουθεί φόρμα “Add / Update

Subcategory" με `OutlinedTextField` και κουμπί, το οποίο είτε δημιουργεί νέα υποκατηγορία (με δημιουργία νέου παιδιού στο `questionsRef.child(selectedSubject).child(name)`), είτε υλοποιεί μετονομασία αντιγράφοντας το υπάρχον υποδένδρο της παλιάς υποκατηγορίας σε νέο όνομα και διαγράφοντας το παλαιό. Με αυτόν τον τρόπο, ο διαχειριστής μπορεί να αναδιατάσσει το περιεχόμενο χωρίς να απαιτείται εκ νέου εισαγωγή όλων των ερωτήσεων.

```
if (selectedSubcategory != null) {
    // Existing questions (read-only cards + edit/delete)
    items(questions) { q ->
        Card(
            modifier = Modifier.fillMaxWidth(),
            elevation = CardDefaults.cardElevation(1.dp)
        ) {
            Column(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(8.dp),
                verticalArrangement = Arrangement.spacedBy(4.dp)
            ) {
                Text(q.questionText, style =
MaterialTheme.typography.titleSmall)
                Text("Options: ${q.options.joinToString(", ")}")
                Text("Correct: ${q.correctAnswer}")
                Row(horizontalArrangement = Arrangement.End, modifier =
Modifier.fillMaxWidth()) {
                    IconButton(onClick = {
                        editingQuestionIndex = questions.indexOf(q)
                        newQuestionText = q.questionText
                        newQuestionOptions = q.options.joinToString(",")
                    }) { Icon(Icons.Filled.Edit, contentDescription = "Edit
Question") }

                    IconButton(onClick = {
                        if (!isPreview) {
                            val questionRef =
questionsRef.child(selectedSubject!!).child(selectedSubcategory!!)
                            questionRef.get().addOnSuccessListener { snap ->
                                val key =
snap.children.elementAtOrNull(questions.indexOf(q))?.key
                                key?.let {
questionRef.child(it).removeValue() }
                                reloadSubjectsAndCurrentBranch()
                            }
                        }
                        questions = questions.filter { it != q }
                    }) { Icon(Icons.Filled.Delete, contentDescription =
"Delete Question") }
                }
            }
        }
    }

    // Editor for add/update question
    item {
        Column(verticalArrangement = Arrangement.spacedBy(8.dp)) {
            HorizontalDivider()
            Text(
                if (editingQuestionIndex != null) "Edit Question" else "Add
Question",
                style = MaterialTheme.typography.titleMedium
            )

            OutlinedTextField(
```

```
                value = newQuestionText,
                onValueChange = { newQuestionText = it },
                label = { Text("Question Text") },
                modifier = Modifier.fillMaxWidth()
            )

            OutlinedTextField(
                value = newQuestionOptions,
                onValueChange = { newQuestionOptions = it },
                label = { Text("Options (comma-separated)") },
                placeholder = { Text("e.g., 1,2,3,4") },
                modifier = Modifier.fillMaxWidth()
            )

            Button(onClick = {
                val text = newQuestionText.trim()
                val options = newQuestionOptions.split(",").map { it.trim()
}.filter { it.isNotEmpty() }
                if (text.isNotEmpty() && options.size >= 2 &&
selectedSubject != null && selectedSubcategory != null) {
                    val subj = selectedSubject!!
                    val sub = selectedSubcategory!!
                    if (!isPreview) {
                        val questionRef =
questionsRef.child(subj).child(sub)
                        if (editingQuestionIndex != null) {
                            // Update existing
                            questionRef.get().addOnSuccessListener { snap ->
                                val key =
snap.children.elementAtOrNull(editingQuestionIndex!!)?.key
                                key?.let {
                                    questionRef.child(it).setValue(
                                        mapOf(
                                            "questionText" to text,
                                            "options" to options,
                                            "correctAnswer" to
options.first()
                                        )
                                    ).addOnSuccessListener {
                                        reloadSubjectsAndCurrentBranch()
                                    }
                                }
                            }
                            questions = questions.mapIndexed { i, old ->
                                if (i == editingQuestionIndex)
Question(text, options, options.first()) else old
                            }
                            editingQuestionIndex = null
                        } else {
                            // Add new
                            questionRef.push().setValue(
                                mapOf(
                                    "questionText" to text,
                                    "options" to options,
                                    "correctAnswer" to options.first()
                                )
                            ).addOnSuccessListener {
```

```
reloadSubjectsAndCurrentBranch() }
                            questions = questions + Question(text, options,
options.first())
                        }
                    } else {
                        // Preview-only local update
                        if (editingQuestionIndex != null) {
                            questions = questions.mapIndexed { i, old ->
                                if (i == editingQuestionIndex)
Question(text, options, options.first()) else old
                            }
                            editingQuestionIndex = null
                        } else {
                            questions = questions + Question(text, options,
options.first())
                        }
                    }
                    newQuestionText = ""
                    newQuestionOptions = ""
                    scope.launch { snackbarHostState.showSnackbar("☑
Question saved") }
                } else {
                    scope.launch { snackbarHostState.showSnackbar("⚠ Enter
question + at least 2 options") }
                }
            }, modifier = Modifier.fillMaxWidth()) {
                Text(if (editingQuestionIndex != null) "Update Question"
else "Add Question")
            }
        }
    }
}
```

30.Διαχείριση ερωτήσεων για επιλεγμένη υποκατηγορία, με δυνατότητες προβολής, επεξεργασίας, διαγραφής και προσθήκης νέων ερωτήσεων μέσω του Admin Dashboard.

Τέλος, όταν υπάρχει επιλεγμένη υποκατηγορία, ενεργοποιείται το τμήμα διαχείρισης ερωτήσεων. Οι ήδη καταχωρισμένες ερωτήσεις εμφανίζονται ως `Card` με το κείμενο της ερώτησης, τις επιλογές και τη σωστή απάντηση. Για κάθε ερώτηση παρέχονται κουμπιά επεξεργασίας και διαγραφής. Η διαγραφή εντοπίζει το αντίστοιχο child στην Realtime Database μέσω του κλειδιού του (Firebase key) και το αφαιρεί, ενώ ταυτόχρονα ενημερώνει τη λίστα `questions`. Η επεξεργασία ρυθμίζει τον δείκτη `editingQuestionIndex` και προγεμίζει τα πεδία `newQuestionText` και `newQuestionOptions`, ώστε ο διαχειριστής να μπορεί να πραγματοποιήσει αλλαγές. Στο κάτω μέρος εμφανίζεται ενότητα με τίτλο “Add Question” ή “Edit Question” ανάλογα με το αν έχει επιλεγεί υπάρχουσα ερώτηση. Ο διαχειριστής εισάγει το κείμενο της ερώτησης και τις επιλογές (χωρισμένες με κόμμα). Με την επιβεβαίωση, η εφαρμογή ελέγχει ότι υπάρχει τουλάχιστον μία ερώτηση και δύο επιλογές και, στη συνέχεια, είτε ενημερώνει την

υπάρχουσα εγγραφή στη βάση (στην περίπτωση edit), είτε δημιουργεί νέα καταχώριση με `push().setValue(...)`. Σε κάθε επιτυχημένη αποθήκευση εμφανίζεται snackbar επιβεβαίωσης, ενώ αποτυχημένες προσπάθειες συνοδεύονται από προειδοποιητικό μήνυμα.

```
private fun uploadQuizDataToFirebase(
    quizData: QuizData,
    subjectsRef: DatabaseReference,
    questionsRef: DatabaseReference
) {
    quizData.subjects.forEach { subject ->
        subjectsRef.push().setValue(subject.name)
        subject.subcategories.forEach { sub ->
            sub.questions.forEach { question ->
                val questionRef =
questionsRef.child(subject.name).child(sub.name).push()
                questionRef.setValue(
                    mapOf(
                        "questionText" to question.questionText,
                        "options" to question.options,
                        "correctAnswer" to question.correctAnswer
                    )
                )
            }
        }
    }
}
```

31.Συνάρτηση μαζικής εισαγωγής δεδομένων κουίζ από αρχείο JSON στο Firebase Realtime Database, με αυτόματη δημιουργία θεμάτων, υποκατηγοριών και ερωτήσεων.

Η βοηθητική συνάρτηση `uploadQuizDataToFirebase` υλοποιεί την προωθημένη λειτουργία μαζικής εισαγωγής quiz από αρχείο JSON. Για κάθε subject στο αντικείμενο `QuizData`, η συνάρτηση καταχωρίζει το όνομα του μαθήματος στον κόμβο `subjects` και, για κάθε υποκατηγορία, δημιουργεί αντίστοιχα children στον κόμβο `questions`, χρησιμοποιώντας `push()` για κάθε ερώτηση. Η δομή που προκύπτει είναι συμβατή με τον τρόπο που η υπόλοιπη εφαρμογή διαβάζει και εμφανίζει τα δεδομένα, επιτρέποντας στον διαχειριστή να δημιουργεί ή να επεκτείνει τράπεζες ερωτήσεων με ελάχιστη χειροκίνητη εργασία.

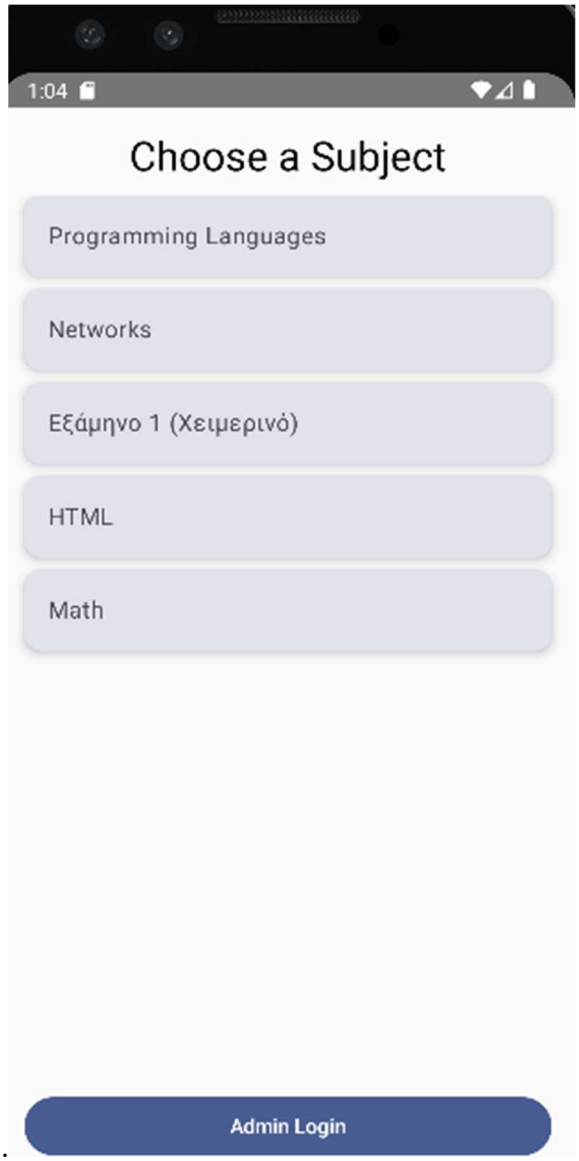
Συνολικά, ο `AdminDashboardScreen` αποτελεί μια ολοκληρωμένη και ευέλικτη κονσόλα διαχείρισης περιεχομένου. Συνδυάζει ελεγχόμενη πρόσβαση μέσω Firebase Authentication, δυναμική ανάκτηση και ενημέρωση δεδομένων μέσω Firebase Realtime Database, υποστήριξη εισαγωγής JSON για μαζική φόρτωση quiz, καθώς και πλούσιες δυνατότητες επεξεργασίας σε επίπεδο μαθημάτων, υποκατηγοριών και ερωτήσεων. Η αρχιτεκτονική του πίνακα διαχειριστή υποστηρίζει τη μελλοντική διεύρυνση της εφαρμογής, τόσο ως προς τον όγκο του εκπαιδευτικού

υλικού όσο και ως προς τις λειτουργικότητες που μπορούν να προστεθούν σε μετέπειτα στάδια ανάπτυξης.

Κεφάλαιο 5 - Περιπτώσεις χρήσης

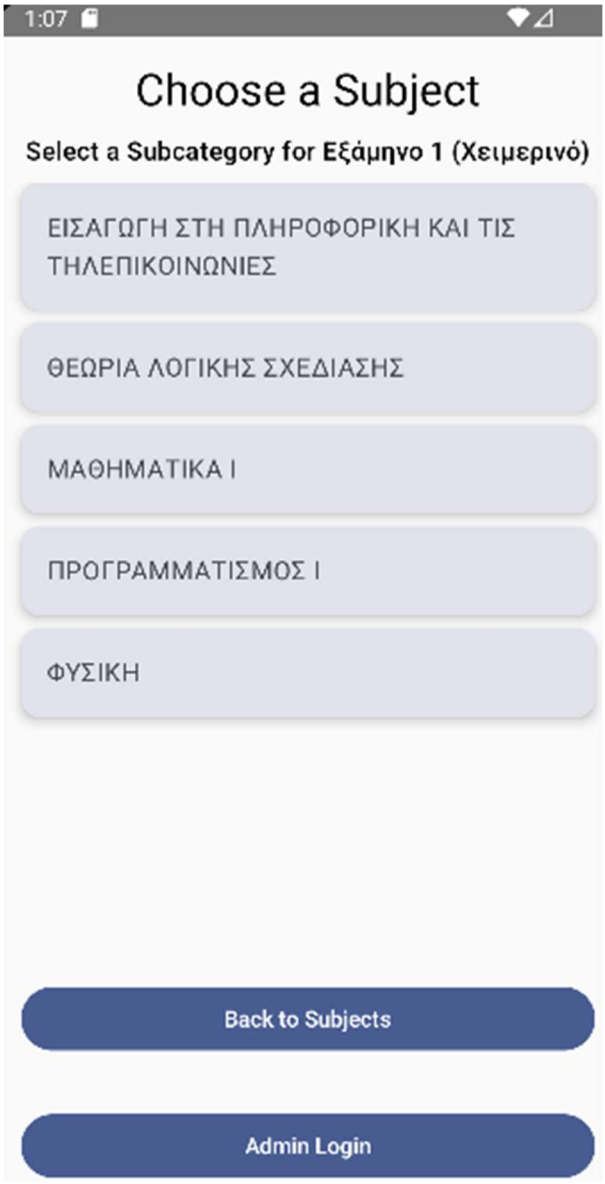
Στο παρόν κεφάλαιο παρουσιάζονται οι κύριες οθόνες και λειτουργίες της εφαρμογής που αναπτύχθηκε στο πλαίσιο της διπλωματικής εργασίας με τίτλο «Σχεδίαση και Ανάπτυξη Εφαρμογής Αυτοαξιολόγησης Φοιτητών».

Στόχος είναι να δοθεί μια ολοκληρωμένη εικόνα της διεπαφής χρήστη και της ροής λειτουργίας της εφαρμογής.



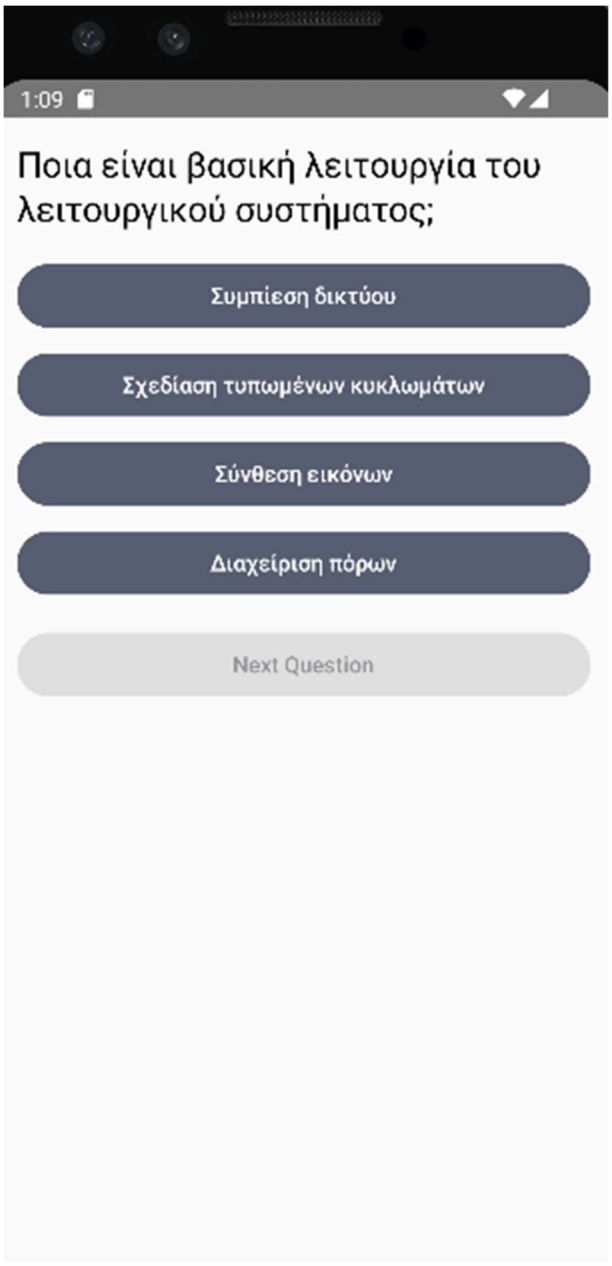
1..Αρχική οθόνη (Home Screen) της εφαρμογής Edutest, όπου ο χρήστης επιλέγει το μάθημα.

Αρχική οθόνη (Home Screen). Ο χρήστης επιλέγει κατηγορία που τον ενδιαφέρει. Εδώ έχει τη δυνατότητα και ο διαχειριστής να επιλέξει να κάνει σύνδεση στην οθόνη Διαχειριστή (Admin Dashboard).



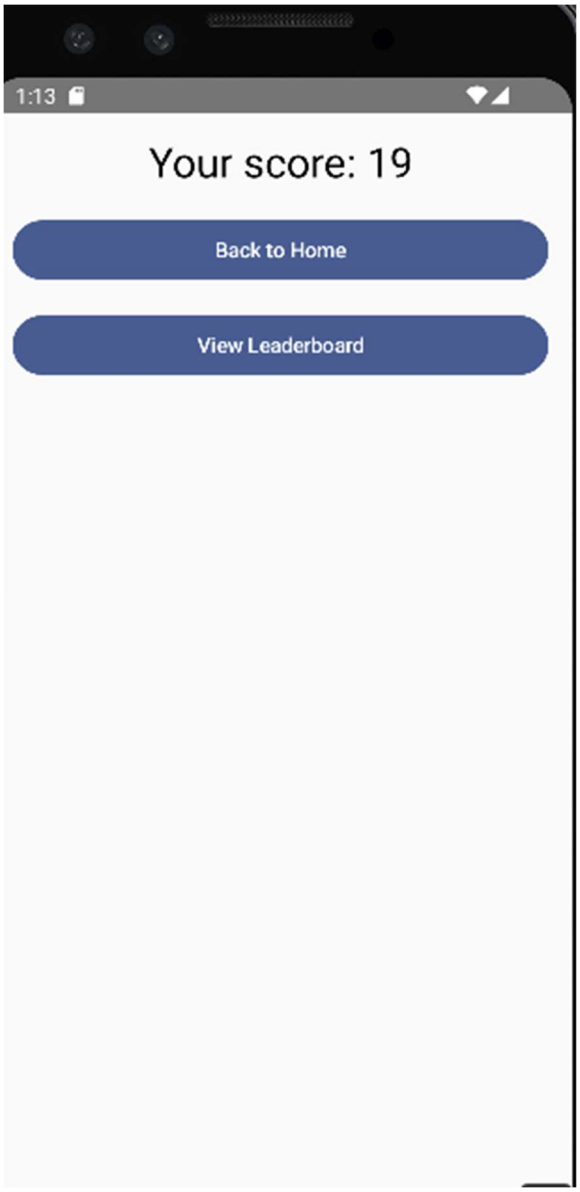
2..Οθόνη επιλογής υποκατηγορίας στην εφαρμογή Edutest, όπου ο χρήστης επιλέγει τη θεματική ενότητα του μαθήματος για την έναρξη του κουίζ.

Οθόνη Υποκατηγοριών. Εδώ ο χρήστης επιλέγει την υποκατηγορία που τον ενδιαφέρει. Έχει τη δυνατότητα να επιστρέψει στην αρχική οθόνη ή να κάνει και σύνδεση ο διαχειριστής.



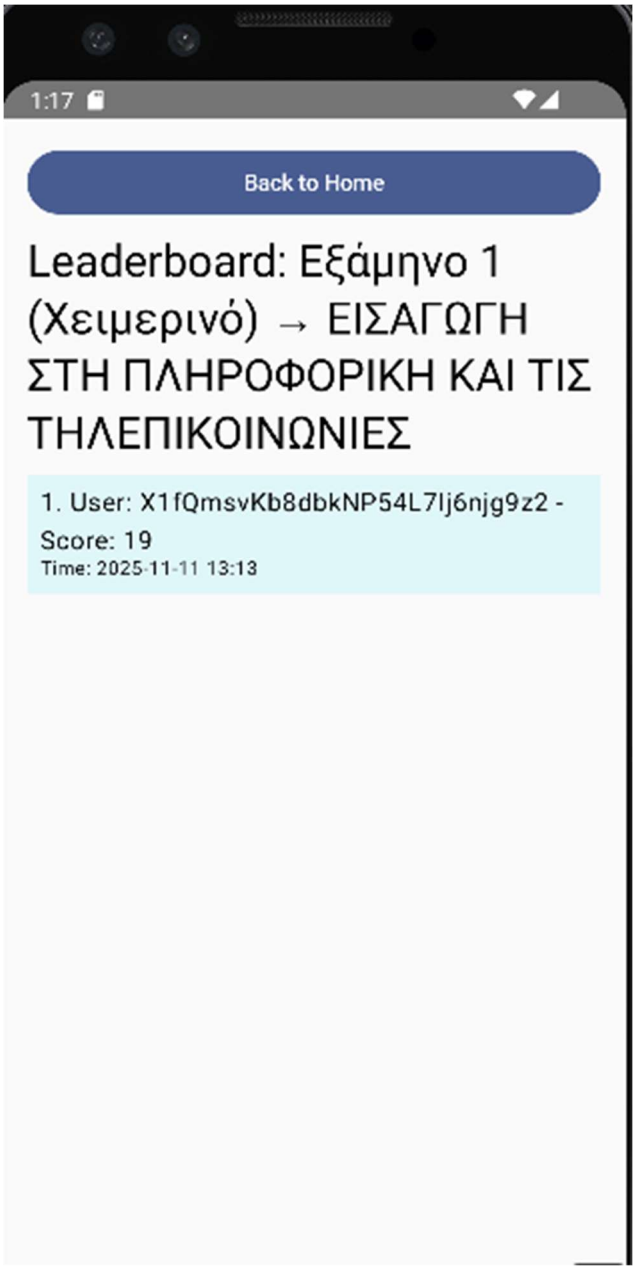
3..Οθόνη διεξαγωγής κουίζ της εφαρμογής Edutest, όπου ο χρήστης απαντά σε ερωτήσεις πολλαπλής επιλογής που ανακτώνται δυναμικά από τη βάση δεδομένων Firebase.

Οθόνη τεστ (Quiz screen). Αφού επιλέξει ο χρήστης υποκατηγορία που τον ενδιαφέρει μεταβαίνει στην οθόνη του τεστ όπου πατά το κουμπί που νομίζει ότι είναι η σωστή απάντηση της ερώτησης και μετά το κουμπί “Next Question”.



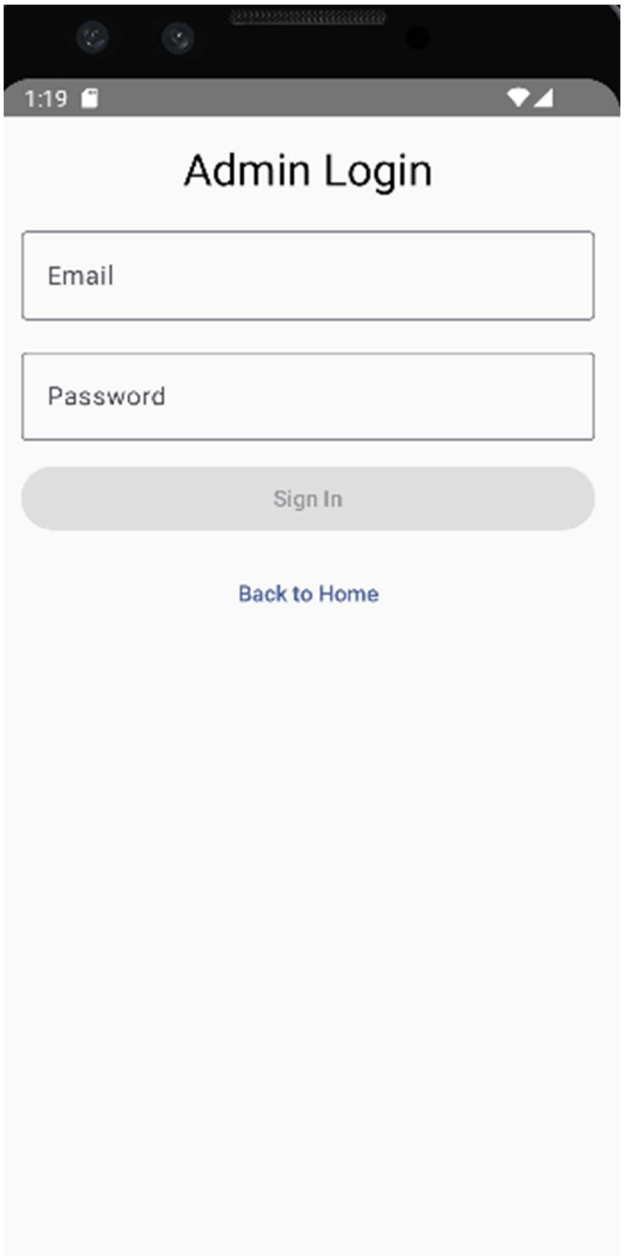
4..Οθόνη αποτελεσμάτων του κουίζ στην εφαρμογή Edutest, η οποία εμφανίζει τη συνολική βαθμολογία του χρήστη και παρέχει επιλογές συνέχισης της πλοήγησης.

Οθόνη αποτελεσμάτων (Results Screen). Εδώ βλέπει ο χρήστης πως πήγε στη διεξαγωγή του τεστ. Μπορεί να επιστρέψει στην αρχική οθόνη ή να μεταβεί στον πίνακα κατάταξης (Leaderboard).



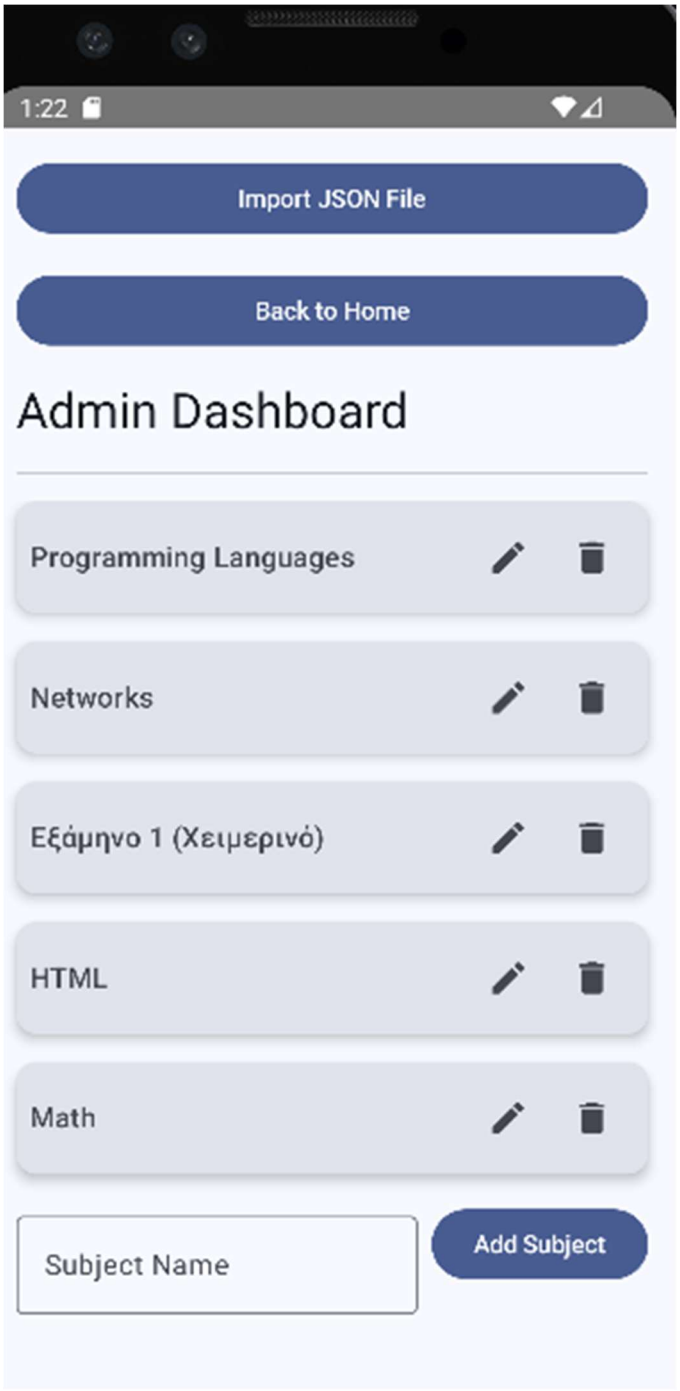
5..Πίνακας κατάταξης (Leaderboard) της εφαρμογής Edutest, όπου παρουσιάζονται οι επιδόσεις των χρηστών ανά μάθημα και υποκατηγορία.

Οθόνη κατάταξης (Leaderboard Screen). Εδώ φαίνεται η κατηγορία και η υποκατηγορία του τεστ που επέλεξε, το σκορ του και η κατάταξη του βάσει άλλων χρηστών. Έχει ένα μοναδικό αναγνωριστικό συσκευής και φαίνεται και η ημερομηνία και η ώρα που διεξήγαγε το τεστ.



6..Οθόνη εισόδου διαχειριστή (Admin Login) της εφαρμογής Edutest, μέσω της οποίας πραγματοποιείται αυθεντικοποίηση με χρήση Firebase Authentication.

Εδώ φαίνεται η οθόνη που βάζει τα διαπιστευτήρια του ο διαχειριστής για να μεταβεί στον πίνακα ελέγχου του Διαχειριστή. Υπάρχει επιλογή να επιστρέψει στην αρχική οθόνη.



7..Πίνακας διαχείρισης (Admin Dashboard) της εφαρμογής Edutest, που επιτρέπει την προσθήκη, επεξεργασία και εισαγωγή εκπαιδευτικού περιεχομένου.

Οθόνη πίνακα ελέγχου διαχειριστή (Admin Dashboard). Εδώ ο διαχειριστής επεξεργάζεται τις κατηγορίες, υποκατηγορίες και ερωτήσεις καθώς υπάρχει και επιλογή για εισαγωγή αρχείου JSON.

Κεφάλαιο 6 — Συμπεράσματα και προτάσεις για μελλοντική ανάπτυξη

6.1 Συνολική αποτίμηση της εργασίας

Η παρούσα εργασία είχε σκοπό τον σχεδιασμό και την ανάπτυξη μιας εφαρμογής με στόχο την αυτοαξιολόγηση των φοιτητών. Χρησιμοποιήθηκαν σύγχρονες τεχνολογίες: η γλώσσα Kotlin, το Jetpack Compose για το γραφικό περιβάλλον και η Realtime Firebase Database για την αποθήκευση και την ανάκτηση των δεδομένων.

Η εφαρμογή επιτρέπει στους χρήστες να διαλέγουν θεματική ενότητα, να απαντούν σε ερωτήσεις πολλαπλής επιλογής και να αυτοαξιολογούν τον εαυτό τους. Ο διαχειριστής έχει τη δυνατότητα να ανανεώνει το περιεχόμενο της εφαρμογής. Επίσης, με την εισαγωγή αρχείου JSON μπορεί να προσθέσει μαζικά περιεχόμενο, χωρίς να απαιτείται επανεκκίνηση της εφαρμογής ή αναβάθμισή της.

Η ανάπτυξη της εφαρμογής επέτρεψε την κατανόηση του τρόπου με τον οποίο μπορούν να συνδυαστούν κινητές συσκευές με τεχνολογίες νέφους για την ανάπτυξη εκπαιδευτικών εφαρμογών. Η σχεδίαση βασίστηκε στην απλότητα και στην εύκολη αναβάθμισή της για μελλοντικές ανάγκες που μπορεί να παρουσιαστούν.

6.2 Συμπεράσματα σχετικά με τη λειτουργικότητα

Η εφαρμογή λειτούργησε με σταθερότητα και αξιοπιστία κατά τη διάρκεια των δοκιμών, παρουσιάζοντας άμεση επικοινωνία με τη βάση δεδομένων και ομαλή πλοήγηση ανάμεσα στις οθόνες.

Η δυνατότητα διαχωρισμού των ερωτήσεων σε θέματα και υποκατηγορίες βοήθησε στην εύκολη και οργανωμένη διαχείριση του περιεχομένου της εφαρμογής.

Η λειτουργία του leaderboard έδωσε μια στοιχειώδη ανταγωνιστικότητα μεταξύ των χρηστών. Η εφαρμογή πέτυχε να έχει εκπαιδευτικό σκοπό και απλή χρήση, κάτι που θεωρείται απαραίτητο για φοιτητές διαφορετικών γνωστικών αντικειμένων.

6.3 Ο ρόλος του διαχειριστή (Admin Panel)

Η ενσωμάτωση του πίνακα ελέγχου διαχειριστή (Admin Dashboard) αποτέλεσε ένα από τα σημαντικότερα σημεία της ανάπτυξης. Η δυνατότητα πρόσβασης μόνο από εξουσιοδοτημένους χρήστες μέσω του Firebase Authentication εξασφαλίζει την ακεραιότητα των δεδομένων.
Η επιλογή εισαγωγής μεγάλου όγκου περιεχομένου μέσω αρχείου JSON καθιστά την εφαρμογή ιδιαίτερα πρακτική. Η συγκεκριμένη δυνατότητα θα μπορούσε να επεκταθεί ώστε να δέχεται και αρχεία CSV ή να διασυνδέεται με άλλες πλατφόρμες εκπαίδευσης.

6.4 Παιδαγωγική αξία της εφαρμογής

Η χρήση ενός συστήματος αυτοαξιολόγησης δίνει τη δυνατότητα στους φοιτητές να παρακολουθούν την πρόοδό τους με διαδραστικό τρόπο σε συγκεκριμένα γνωστικά αντικείμενα που τους ενδιαφέρουν. Η αμεσότητα της ανατροφοδότησης ενισχύει την κατανόηση και τη διατήρηση της γνώσης, ενώ ταυτόχρονα καλλιεργεί τη δεξιότητα της αυτορρύθμισης της μάθησης. Σε σύγκριση με τα παραδοσιακά μέσα αξιολόγησης, η εφαρμογή αυτή προσφέρει ευελιξία, αυτονομία και προσβασιμότητα, επιτρέποντας στους χρήστες να αξιολογούν τις γνώσεις τους οποιαδήποτε στιγμή και από οποιαδήποτε συσκευή. Η απουσία ανάγκης φυσικής παρουσίας την καθιστά κατάλληλη για σύγχρονες μορφές εκπαίδευσης, όπως η εξ αποστάσεως μάθηση και η μικτή μάθηση.

6.5 Περιορισμοί της εφαρμογής

Παρά τα θετικά αποτελέσματα, η εφαρμογή παρουσιάζει ορισμένους περιορισμούς που αξίζει να επισημανθούν.
Αρχικά, η εφαρμογή εξαρτάται από τη σύνδεση στο διαδίκτυο, με ό,τι αυτό συνεπάγεται, όπως περιοχές με χαμηλή συνδεσιμότητα. Επίσης, δεν περιλαμβάνει δυνατότητα προσωπικής αποθήκευσης δεδομένων (offline caching), η οποία θα επέτρεπε τη χρήση χωρίς σύνδεση.

Άλλος περιορισμός είναι η απουσία προηγμένων στατιστικών εργαλείων που θα μπορούσαν να προσφέρουν πιο ολοκληρωμένη εικόνα της προόδου των φοιτητών.

6.6 Προτάσεις για μελλοντική ανάπτυξη

Η παρούσα εφαρμογή μπορεί να αποτελέσει τη βάση για ευρύτερη ανάπτυξη και ενσωμάτωση πρόσθετων δυνατοτήτων.

Ενδεικτικά, προτείνονται τα εξής βήματα:

1. Προσθήκη λειτουργίας offline χρήσης, ώστε να μπορούν οι χρήστες να χρησιμοποιούν την εφαρμογή χωρίς σύνδεση στο διαδίκτυο.
2. Δημιουργία προφίλ χρήστη με ιστορικό επιδόσεων και στατιστικών προόδου.
3. Εισαγωγή συστήματος ειδοποιήσεων που θα υπενθυμίζει στον χρήστη να επαναλάβει το τεστ ή να δοκιμάσει άλλα γνωστικά αντικείμενα.
4. Ενσωμάτωση πολυμεσικού περιεχομένου για πιο σύνθετες μορφές ερωτήσεων.
5. Προσθήκη πολλαπλών ρόλων χρηστών, όπως καθηγητές και βοηθοί, για τη διαχείριση του περιεχομένου.
6. Εξαγωγή αποτελεσμάτων σε μορφή αναφορών (PDF) ώστε να μπορούν να αποστέλλονται, για παράδειγμα, μέσω ηλεκτρονικού ταχυδρομείου.
7. Σύνδεση με άλλες πλατφόρμες e-learning, όπως το Moodle.

Η επέκταση αυτών των λειτουργιών θα ενισχύσει σημαντικά τη χρηστικότητα και τη βιωσιμότητα της εφαρμογής, καθιστώντας την ένα ολοκληρωμένο εκπαιδευτικό εργαλείο σε ακαδημαϊκά ή επαγγελματικά περιβάλλοντα.

6.7 Επίλογος

Η εργασία αυτή απέδειξε ότι είναι εφικτό να δημιουργηθεί ένα πλήρως λειτουργικό και δυναμικό σύστημα αυτοαξιολόγησης με τη χρήση ελεύθερων εργαλείων και σύγχρονων τεχνολογιών.

Η εφαρμογή μπορεί να αποτελέσει τη βάση για περαιτέρω έρευνα και ανάπτυξη σε θέματα ψηφιακής μάθησης, εκπαιδευτικών τεχνολογιών και εξατομικευμένης αξιολόγησης.

Η σύνδεση της εκπαίδευσης με τις τεχνολογίες κινητών συσκευών ανοίγει νέους δρόμους στη μαθησιακή διαδικασία, προωθώντας την αυτονομία, την άμεση ανατροφοδότηση και τη συνεχή βελτίωση των δεξιοτήτων.

Η εμπειρία από την ανάπτυξη αυτής της εφαρμογής ανέδειξε τη σημασία του σωστού σχεδιασμού, της απλότητας στη διεπαφή και της δυνατότητας προσαρμογής σε διαφορετικά εκπαιδευτικά περιβάλλοντα.

Βιβλιογραφία

1. Google Developers. *Firebase Documentation.* Διαθέσιμο στο: https://firebase.google.com/docs (Πρόσβαση: 2025).
2. Android Developers. *Jetpack Compose Documentation.* Διαθέσιμο στο: https://developer.android.com/jetpack/compose.
3. Android Developers. *Kotlin for Android Developers.* Διαθέσιμο στο: https://developer.android.com/kotlin.
4. W3Schools. *Firebase Realtime Database Tutorial.* Διαθέσιμο στο: https://www.w3schools.com/firebase/firebase_database.asp.
5. Nielsen, J. (1993). *Usability Engineering.* Academic Press, Boston.
6. Churcher, K. (2018). *Gamification in Learning: Applications of Interactive Digital Games in Education.* IGI Global.
7. Prensky, M. (2001). *Digital Natives, Digital Immigrants.* On the Horizon, 9(5), 1–6.
8. Ally, M. (Ed.). (2009). *Mobile Learning: Transforming the Delivery of Education and Training.* Athabasca University Press.
9. Traxler, J. (2009). *Learning in a Mobile Age.* International Journal of Mobile and Blended Learning, 1(1), 1–12.
10. Sharples, M., Taylor, J., & Vavoula, G. (2007). *A Theory of Learning for the Mobile Age.* In R. Andrews & C. Haythornthwaite (Eds.), *The SAGE Handbook of E-learning Research.* Sage Publications.
11. ISO 9241-210:2019. *Ergonomics of Human-System Interaction – Human-Centred Design for Interactive Systems.* International Organization for Standardization.
12. Creswell, J. W. (2014). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches.* SAGE Publications.
13. Statista Research Department (2024). *Global Mobile Learning Market Size.* Διαθέσιμο στο: https://www.statista.com.

Παράρτημα Α — Ενδεικτικός Κώδικας

Α.1 Παράδειγμα φόρτωσης δεδομένων από Firebase

```
val questionsRef = FirebaseDatabase.getInstance()
    .getReference("questions")
    .child(subject)
    .child(subcategory)

questionsRef.get().addOnSuccessListener { snapshot ->
    val loadedQuestions = snapshot.children.mapNotNull { child ->
        val text = child.child("questionText").value as? String
        val       options       =       (child.child("options").value       as?
List<*>)?.filterIsInstance<String>()
        val correct = child.child("correctAnswer").value as? String
        if (text != null && options != null && correct != null)
            Question(text, options, correct)
        else null
    }
}
```

Α.2 Παράδειγμα δομής αρχείου JSON

```
{
  "subjects": [
    {
      "name": "Programming Languages",
      "subcategories": [
        {
          "name": "Java",
          "questions": [
            {
              "questionText": "What keyword is used to define a class in Java?",
              "options": ["class", "def", "function", "struct"],
              "correctAnswer": "class"
            }
          ]
        }
      ]
    },
    {
```

```
      "name": "Networks",
      "subcategories": [
        {
          "name": "General",
          "questions": [
            {
              "questionText": "What does LAN stand for?",
              "options": ["Local Area Network", "Large Area Network", "Long
Access Node", "Light Access Network"],
              "correctAnswer": "Local Area Network"
            }
          ]
        }
      ]
    }
  ]
}
```

Παράρτημα Β — Αρχιτεκτονική και Τεχνικά Χαρακτηριστικά

- **Γλώσσα προγραμματισμού:** Kotlin
- **Περιβάλλον ανάπτυξης:** Android Studio
- **UI Framework:** Jetpack Compose
- **Υπηρεσίες Backend:** Firebase Authentication, Firebase Realtime Database
- **Στόχος Android SDK:** 34
- **Συμβατότητα:** Android 8.0 (API 26) και άνω
- **Τύπος βάσης δεδομένων:** NoSQL (Firebase Realtime Database)
- **Μορφή αποθήκευσης ερωτήσεων:** JSON

Παράρτημα Γ — Αποτελέσματα Δοκιμών

Στον παρακάτω πίνακα παρουσιάζονται συνοπτικά τα αποτελέσματα των δοκιμών που πραγματοποιήθηκαν:

Σενάριο Δοκιμής	Περιγραφή	Αποτέλεσμα	Παρατηρήσεις
Σύνδεση Διαχειριστή	Έλεγχος πρόσβασης με σωστά στοιχεία	Επιτυχής	Ο διαχειριστής οδηγείται στο Dashboard

Σενάριο Δοκιμής	Περιγραφή	Αποτέλεσμα	Παρατηρήσεις
Εισαγωγή JSON	Ανέβασμα αρχείου με νέα ερωτήματα	Επιτυχής	Τα δεδομένα ενημερώθηκαν στη βάση
Εκτέλεση Τεστ	Απάντηση σε 10 ερωτήσεις	Επιτυχής	Εμφανίστηκε σωστά το τελικό σκορ
Προβολή Leaderboard	Εμφάνιση κατάταξης	Επιτυχής	Εμφανίστηκε σωστά ο χρήστης
Offline λειτουργία	Χωρίς σύνδεση στο διαδίκτυο	Μερική	Δεν φορτώνονται ερωτήσεις χωρίς σύνδεση

Παράρτημα Δ— Τεχνικός Οδηγός Εγκατάστασης

Δ.1 Εισαγωγή

Το παρόν παράρτημα περιγράφει αναλυτικά τα βήματα που απαιτούνται για την εγκατάσταση, ρύθμιση και εκτέλεση της εφαρμογής που αναπτύχθηκε στο πλαίσιο της παρούσας διπλωματικής εργασίας.
Ο οδηγός αυτός απευθύνεται τόσο σε προγραμματιστές που επιθυμούν να επεκτείνουν την εφαρμογή, όσο και σε εκπαιδευτικά ιδρύματα που θέλουν να την αξιοποιήσουν σε πραγματικό περιβάλλον.

Δ.2 Προαπαιτούμενα

Για την επιτυχή εκτέλεση της εφαρμογής, απαιτούνται τα παρακάτω:

- **Λειτουργικό σύστημα:** Windows 10 ή νεότερο / macOS / Linux
- **Android Studio:** Έκδοση Flamingo (ή νεότερη)
- **Java Development Kit (JDK):** Έκδοση 17 ή νεότερη
- **Ενεργός λογαριασμός Google** για τη χρήση των υπηρεσιών **Firebase**
- **Σύνδεση στο διαδίκτυο** για τη φόρτωση δεδομένων από τη βάση
- **Android Emulator** ή πραγματική συσκευή με Android 8.0 (API 26) ή νεότερη

Δ.3 Βήματα Εγκατάστασης

Δ.3.1 Λήψη του έργου

1. Κατεβάστε τον φάκελο του έργου από το αποθετήριο Git
2. Αποσυμπιέστε το αρχείο και ανοίξτε τον φάκελο **Meta** από το Android Studio.
3. Επιλέξτε **File → Open** και πλοηγηθείτε στον φάκελο του έργου.
4. Περιμένετε να ολοκληρωθεί η διαδικασία συγχρονισμού των εξαρτήσεων Gradle.

Δ.3.2 Σύνδεση με Firebase

1. Μεταβείτε στο https://console.firebase.google.com
2. Δημιουργήστε ένα νέο Project (π.χ. *MetaApp*).
3. Επιλέξτε **Add App → Android** και εισαγάγετε το **package name** της εφαρμογής, π.χ. `com.example.meta`.
4. Κατεβάστε το αρχείο **google-services.json** και τοποθετήστε το στον φάκελο `app/` του έργου.
5. Ενεργοποιήστε στο Firebase:
   - **Authentication → Email/Password**
   - **Realtime Database**
6. Ρυθμίστε τη βάση σε **Production mode** για ασφάλεια.
7. Αντιγράψτε τον εξής κόμβο για τον διαχειριστή:

```
admins
  <UID_του_admin_χρήστη>: true
```

Δ.3.3 Εκτέλεση της Εφαρμογής

1. Ανοίξτε το Android Studio και επιλέξτε **Run → Run 'app'**.
2. Επιλέξτε έναν emulator ή συνδέστε μια φυσική Android συσκευή.
3. Περιμένετε να ολοκληρωθεί η διαδικασία εγκατάστασης.
4. Μετά την εκκίνηση, η εφαρμογή θα φορτώσει την αρχική οθόνη. Ο χρήστης μπορεί να:
   - Επιλέξει μάθημα → Υποκατηγορία → Τεστ
   - Προβάλει αποτελέσματα και leaderboard
   - Αν είναι διαχειριστής, να εισέλθει στο **Admin Dashboard**

Δ.4 Ενημέρωση Περιεχομένου μέσω JSON

Η εφαρμογή υποστηρίζει εισαγωγή νέων μαθημάτων και ερωτήσεων από αρχείο **JSON** μέσω του πίνακα διαχειριστή.

Διαδικασία:

1. Συνδεθείτε ως διαχειριστής.
2. Επιλέξτε Import JSON File.
3. Επιλέξτε το αρχείο (π.χ. quiz_data.json) από τον φάκελο Downloads ή Documents.

Το σύστημα προσθέτει τα νέα δεδομένα χωρίς να διαγράφει τα υπάρχοντα

Δ.5 Δομή Δεδομένων Firebase

Η εφαρμογή χρησιμοποιεί τη βάση Firebase Realtime Database με την εξής ιεραρχία:

```
subjects/
  Math: "Math"
  HTML: "HTML"
  Networks: "Networks"

questions/
  Math/
    Algebra/
      q1/
        questionText: ...
        options: [...]
        correctAnswer: ...
scores/
  <device_id>/
    Math/
      Algebra/
        {score, timestamp}
admins/
  <UID>: true
```

Δ.6 Αντιμετώπιση Προβλημάτων

Πρόβλημα	**Πιθανή Αιτία**	**Λύση**
Δεν φορτώνονται δεδομένα	Δεν υπάρχει σύνδεση στο διαδίκτυο ή λανθασμένο `google-services.json`	Ελέγξτε τη σύνδεση και επανεισάγετε το αρχείο

Πρόβλημα	Πιθανή Αιτία	Λύση
Δεν εμφανίζονται υποκατηγορίες	Ελλιπή δεδομένα στο Firebase	Ελέγξτε τη δομή JSON
Δεν επιτρέπεται πρόσβαση στο Dashboard	Δεν έχει οριστεί admin UID	Προσθέστε το UID του χρήστη στον κόμβο `admins`
Το app δεν εγκαθίσταται	Ελλιπής SDK ή λανθασμένη Gradle config	Εκτελέστε **File → Sync Project with Gradle Files**

Δ.7 Συμπεράσματα

Ο τεχνικός οδηγός επιβεβαιώνει ότι η εφαρμογή μπορεί να εγκατασταθεί και να λειτουργήσει ομαλά τόσο σε **τοπικό περιβάλλον ανάπτυξης όσο και σε πραγματικές Android συσκευές.** Η χρήση του Firebase προσφέρει ευκολία διαχείρισης δεδομένων, ασφάλεια και δυνατότητα επεκτασιμότητας για μελλοντικές εκδόσεις.