Πανεπιστήμιο Πελοποννήσου
Τμήμα Πληροφορικής και Τηλεπικοινωνίων

Σχεδίαση και Ανάπτυξη Εφαρμογής Κινητού για την Οργάνωση και Υποστήριξη της Διοίκησης Σχολικής Μονάδας

Βεζονιαράκης Δημήτρης

Α.Μ.: 2022 2023 02002

Επιβλέπων: Νίκος Τσελίκας – Καθηγητής

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

Ιούνιος 2025

University of Peloponnese
Department of Information and Telecommunications

Design and Development of a Mobile Application to Support the Administrative Functions of School Leadership

Vezoniarakis Dimitris

2022 2023 02002

Supervisor: Nikolaos Tselikas - Professor

Diploma thesis for M.Sc. program in “Computer Science”

June 2025

Abstract

Title: Design and Development of a Mobile Application to Support the Administrative Functions of School Leadership

This thesis focuses on the design and development of a mobile application intended to support the administrative responsibilities of school principals. School leadership today encompasses a wide range of demanding tasks, including scheduling, document creation, staff leave management, incident recording, and communication with various stakeholders. Despite the importance of these responsibilities, there is currently no modern, integrated mobile solution that enables principals to manage these functions efficiently and on the go.

The proposed application was developed using Flutter to enable cross-platform compatibility and was connected to a backend database using either Firebase or a RESTful API. This study documents the analysis of real-world user requirements, the definition of functional and non-functional specifications, the creation of UI mockups, and the implementation of core features, followed by a pilot evaluation in a real school environment. The final product aims to enhance the principal's autonomy and efficiency by minimizing bureaucracy and improving the organization of administrative tasks within the school unit.

Keywords :

Flutter, Firebase, Cloud Firestore, Google Drive Integration, Template-based Document Generation, Firestore CRUD, Web App Script, Form Validation, Dynamic Dropdown, Document Export, Material Design UI, Mobile App for School Administration, Google Docs Automation, Dynamic Template Selection, Real-time Firestore Sync, Google Apps Script Export, Document Management in Flutter.

Περίληψη

Τίτλος: Σχεδίαση και Ανάπτυξη Εφαρμογής Κινητού για την Υποστήριξη των Διοικητικών Λειτουργιών της Σχολικής Ηγεσίας

Η παρούσα διπλωματική εργασία πραγματεύεται τον σχεδιασμό και την ανάπτυξη μιας εφαρμογής κινητού τηλεφώνου, με στόχο την υποστήριξη των διοικητικών λειτουργιών του Διευθυντή σχολικής μονάδας. Ο ρόλος της σχολικής ηγεσίας έχει καταστεί ιδιαίτερα απαιτητικός, καθώς περιλαμβάνει καθημερινή διαχείριση ραντεβού, εγγράφων, αδειών προσωπικού, συμβάντων και επικοινωνίας με ποικίλους φορείς. Παρά τη σπουδαιότητα του έργου του Διευθυντή, απουσιάζει ένα σύγχρονο ψηφιακό εργαλείο που να υποστηρίζει κεντρικά και φορητά όλες τις κρίσιμες διοικητικές διαδικασίες.

Η εφαρμογή υλοποιήθηκε με χρήση του Flutter, για την ανάπτυξη πολυπλατφορμικού περιβάλλοντος διεπαφής, και διασυνδέθηκε με βάση δεδομένων μέσω Firebase ή RESTful API. Η εργασία περιλαμβάνει την ανάλυση των πραγματικών απαιτήσεων των χρηστών, τον καθορισμό λειτουργικών και μη λειτουργικών χαρακτηριστικών, τη δημιουργία mockups, την υλοποίηση των βασικών λειτουργιών και την πιλοτική δοκιμή σε πραγματικό σχολικό πλαίσιο. Το τελικό προϊόν φιλοδοξεί να ενισχύσει την αποτελεσματικότητα και αυτονομία του Διευθυντή, μειώνοντας τη γραφειοκρατία και προάγοντας την οργάνωση της σχολικής διοίκησης.

Λέξεις-Κλειδιά (Keywords):

Flutter, Firebase, Cloud Firestore, Ενσωμάτωση με Google Drive, Δημιουργία Εγγράφων βάσει Προτύπων, CRUD στο Firestore, Google Apps Script, Επικύρωση Φόρμας, Δυναμική Επιλογή από Λίστα, Εξαγωγή Εγγράφων, Διεπαφή Χρήστη με Υλικό Σχεδιασμό (Material Design), Εφαρμογή για Κινητά στη Διοίκηση Σχολείου, Αυτοματισμός Εγγράφων Google, Δυναμική Επιλογή Προτύπων, Συγχρονισμός Firestore σε Πραγματικό Χρόνο, Εξαγωγή με Google Apps Script, Διαχείριση Εγγράφων στο Flutter.

Εγκατάσταση

Εφαρμογή κινητού (android) για τη διαχείριση ραντεβού, εγγράφων, βιβλίο συμβάντων, από τη Διεύθυνση Σχολικής Μονάδας. Η εφαρμογή υλοποιήθηκε στο πλαίσιο της Διπλωματικής μου Εργασίας του Προγράμματος Μεταπτυχιακών Σπουδών στην Επιστήμη Υπολογιστών, στο Πανεπιστήμιο Πελοποννήσου (Ιούνιος 2025).

Οδηγίες εγκατάστασης

Βήμα 1: Ενεργοποίηση Άγνωστων Πηγών: Πηγαίνετε στις Ρυθμίσεις → Ασφάλεια
Ενεργοποιήστε το: Άγνωστες Πηγές (Allow installation from unknown sources)
Απαιτείται για εγκατάσταση εφαρμογών που δεν βρίσκονται στο Play Store.

Βήμα 2: Σκανάρετε το QR Code.

Βήμα 3: Επιλέξετε « Άνοιγμα στο Πρόγραμμα Περιήγησης».

Βήμα 4: Επιβεβαίωση για λήψη.
Μπορεί να εμφανιστεί μήνυμα "Αυτή η εφαρμογή μπορεί να είναι επιβλαβής".
Πατήστε "Εγκατάσταση ούτως ή άλλως" (Install anyway).

Βήμα 5: Άνοιγμα.

https://github.com/dvezon/myappreleases/releases/download/V1.0.0/apprelease.apk

Κώδικας Εφαρμογής

https://github.com/dvezon/myfdb/releases/tag/V1.0

Περιεχόμενα

Παραρτήματα (Appendix)

Εισαγωγή

Η παρούσα διπλωματική εργασία εξετάζει τη διαδικασία σχεδιασμού και ανάπτυξης μιας εφαρμογής κινητού τηλεφώνου που προορίζεται να προσφέρει γραμματειακή και διοικητική υποστήριξη στον Διευθυντή ενός Γυμνασίου. Οι ανάγκες του Διευθυντή είναι πολυδιάστατες και απαιτούν ένα ευέλικτο, αποτελεσματικό εργαλείο που να καλύπτει ένα ευρύ φάσμα διοικητικών και οργανωτικών λειτουργιών.

Αντικείμενο της εργασίας

Ο Διευθυντής σε μία σχολική μονάδα αποτελεί το διοικητικό και παιδαγωγικό κέντρο αυτής και καλείται να ανταποκριθεί καθημερινά σε ένα ευρύ φάσμα αρμοδιοτήτων, όπως η οργάνωση του σχολικού προγράμματος, η διαχείριση εγγράφων, η επίλυση προβλημάτων και η επικοινωνία με εκπαιδευτικούς, γονείς και υπηρεσίες.

Στο πλαίσιο αυτό, προτείνεται η ανάπτυξη μιας πολυλειτουργικής, εύχρηστης και φορητής εφαρμογής που να καλύπτει βασικές διοικητικές ανάγκες του Διευθυντή, όπως:

- τον προγραμματισμό και τη διαχείριση ραντεβού και υπενθυμίσεων,
- τη διαχείριση αδειών εκπαιδευτικών,
- τη δημιουργία και διαχείριση εγγράφων,
- την καταγραφή και παρακολούθηση συμβάντων που αφορούν τη σχολική μονάδα.

Η εφαρμογή αναπτύσσεται με στόχο να συμβάλει στη μείωση της γραφειοκρατίας και στην ψηφιακή αναβάθμιση της σχολικής διοίκησης, προσφέροντας στον Διευθυντή ένα σύγχρονο εργαλείο άμεσης πρόσβασης και οργάνωσης του έργου του μέσω κινητής συσκευής.

Σκοπός και στόχοι

Σκοπός της παρούσας διπλωματικής εργασίας είναι η δημιουργία μιας σύγχρονης, φορητής και λειτουργικά επαρκούς εφαρμογής κινητού τηλεφώνου που να υποστηρίζει ουσιαστικά το έργο του Διευθυντή Γυμνασίου σε επίπεδο γραμματειακής και διοικητικής οργάνωσης. Η εφαρμογή αυτή φιλοδοξεί να αποτελέσει ένα εργαλείο καθημερινής χρήσης, το οποίο θα διευκολύνει τον προγραμματισμό, την επικοινωνία, τη λήψη αποφάσεων και τη διαχείριση πληροφοριών μέσα στο σχολικό περιβάλλον.

Για την επίτευξη του παραπάνω σκοπού, τίθενται οι εξής επιμέρους στόχοι:

- Καταγραφή αναγκών του Διευθυντή σε σχέση με τα καθημερινά του καθήκοντα και τις διοικητικές διαδικασίες.
- Σχεδίαση και υλοποίηση μιας εφαρμογής με φιλικό και λειτουργικό περιβάλλον χρήστη (UI), κατάλληλο για χρήση από άτομα με περιορισμένο τεχνολογικό υπόβαθρο.
- Ανάπτυξη βασικών λειτουργιών, όπως η διαχείριση ραντεβού, υπενθυμίσεων, εγγράφων, αδειών και συμβάντων.
- Ενσωμάτωση τεχνολογιών cloud για την ασφαλή αποθήκευση και συγχρονισμό δεδομένων.

- Πιλοτική εφαρμογή και αξιολόγηση της χρήσης της εφαρμογής σε πραγματικό σχολικό περιβάλλον.
- Καταγραφή παρατηρήσεων και προτάσεων για μελλοντική εξέλιξη της εφαρμογής και πιθανή επέκτασή της σε άλλες βαθμίδες ή ρόλους της εκπαιδευτικής διοίκησης.

Η εργασία συνδυάζει τις τεχνικές γνώσεις ανάπτυξης λογισμικού με τη διοικητική πραγματικότητα του σχολείου, ώστε να προκύψει ένα χρήσιμο και αποτελεσματικό ψηφιακό εργαλείο.

Μεθοδολογία

Η μεθοδολογία που ακολουθήθηκε στην παρούσα εργασία συνδυάζει στοιχεία ανάλυσης απαιτήσεων, σχεδίασης και υλοποίησης λογισμικού, με τεχνικές αξιολόγησης της χρηστικότητας της εφαρμογής σε πραγματικό περιβάλλον. Η ανάπτυξη της εφαρμογής στηρίχθηκε σε σύγχρονες πρακτικές λογισμικού και κύκλους επαναληπτικής βελτίωσης (iterative development), ακολουθώντας κατά βάση τη μεθοδολογία Agile.

Τα βασικά στάδια της μεθοδολογικής προσέγγισης περιλαμβάνουν:

Καταγραφή αναγκών και λειτουργικών απαιτήσεων του Διευθυντή Γυμνασίου, μέσω παρατήρησης, ημι-δομημένων συνεντεύξεων και ανάλυσης της υπάρχουσας πρακτικής.

Σχεδίαση της εφαρμογής, τόσο σε επίπεδο αρχιτεκτονικής (δομής δεδομένων, διαχείρισης ροών, σύνδεσης με backend) όσο και σε επίπεδο διεπαφής χρήστη (UI/UX), με χρήση κατάλληλων εργαλείων σχεδίασης (π.χ. Figma).

Υλοποίηση της εφαρμογής με τη χρήση της τεχνολογίας Flutter, ώστε να είναι διαθέσιμη σε Android και iOS, και διασύνδεσή της με πλατφόρμα αποθήκευσης δεδομένων (Firebase ή RESTful API).

Δοκιμές λειτουργικότητας, με έμφαση στην ορθή λειτουργία των βασικών δυνατοτήτων: ραντεβού, υπενθυμίσεις, έγγραφα, άδειες και ημερολόγιο συμβάντων.

Πιλοτική χρήση της εφαρμογής από Διευθυντή σχολικής μονάδας για προκαθορισμένο χρονικό διάστημα και συλλογή ανατροφοδότησης.

Αξιολόγηση αποτελεσμάτων και καταγραφή προτάσεων για μελλοντική επέκταση ή βελτίωση.

Η μεθοδολογία αυτή επιλέχθηκε επειδή εξασφαλίζει ευελιξία, προσαρμογή στις πραγματικές ανάγκες του χρήστη και δυνατότητα ενσωμάτωσης ανατροφοδότησης σε επόμενες εκδόσεις του λογισμικού.

Δομή της εργασίας

Η εργασία οργανώνεται σε επτά βασικά κεφάλαια, καθένα από τα οποία συμβάλλει στην πληρέστερη τεκμηρίωση της διαδικασίας σχεδιασμού, υλοποίησης και αξιολόγησης της εφαρμογής.

Κεφάλαια :

1. Περιλαμβάνει την εισαγωγή στο θέμα, τον καθορισμό του αντικειμένου, τον σκοπό και τους στόχους της εργασίας, τη μεθοδολογία που ακολουθήθηκε και τη δομή του κειμένου.
2. Παρουσιάζει το θεωρητικό υπόβαθρο και τη σχετική βιβλιογραφία που αφορά τη σχολική διοίκηση, την ανάγκη για ψηφιακά εργαλεία και τις τεχνολογίες που επιλέχθηκαν για την ανάπτυξη της εφαρμογής.
3. Εστιάζει στην καταγραφή των αναγκών και των απαιτήσεων του Διευθυντή, όπως αυτές προκύπτουν από την καθημερινή του εμπειρία.
4. Περιγράφει τη σχεδίαση της εφαρμογής, τόσο σε επίπεδο αρχιτεκτονικής και λειτουργιών, όσο και σε επίπεδο διεπαφής χρήστη.
5. Αναφέρεται στην υλοποίηση της εφαρμογής με χρήση Flutter και Firebase ή RESTful API, και τεκμηριώνει τις κύριες λειτουργίες της.
6. Παρουσιάζει την πιλοτική εφαρμογή στο σχολικό περιβάλλον, τη μεθοδολογία αξιολόγησης και τα αποτελέσματα που προέκυψαν.
7. Συνοψίζει τα συμπεράσματα της εργασίας, αναδεικνύει τα οφέλη της εφαρμογής και προτείνει πιθανές κατευθύνσεις για μελλοντική ανάπτυξη.

Στο τέλος της εργασίας περιλαμβάνονται παραρτήματα με τεχνικά διαγράμματα και δείγματα κώδικα που χρησιμοποιήθηκαν κατά τη διάρκεια του έργου.

Θεωρητικό Υπόβαθρο και Συναφές Έργο

Ο ρόλος του Διευθυντή στη σχολική διοίκηση

Ο Διευθυντής Γυμνασίου αποτελεί τον βασικό συντονιστή της σχολικής μονάδας και φέρει την ευθύνη για την ομαλή λειτουργία της. Εκτός από τον παιδαγωγικό του ρόλο, επιτελεί πλήθος διοικητικών καθηκόντων, όπως:

- τη διαχείριση εγγράφων και αλληλογραφίας,
- την οργάνωση συναντήσεων με εκπαιδευτικούς, γονείς ή υπηρεσίες,
- την έγκριση αδειών προσωπικού,
- την παρακολούθηση της καθημερινής λειτουργίας της μονάδας,
- τη λήψη και τεκμηρίωση αποφάσεων,
- και την επικοινωνία με τις Διευθύνσεις Εκπαίδευσης.

Η έλλειψη εξειδικευμένων εργαλείων υποστήριξης οδηγεί συχνά σε χρήση πολλαπλών ανεπίσημων μέσων (π.χ. χαρτιά, excel, προσωπικά ημερολόγια), γεγονός που δυσχεραίνει την οργάνωση και την αποδοτικότητα.

Ψηφιακά εργαλεία στη σχολική διοίκηση

Η αξιοποίηση ψηφιακών τεχνολογιών στη σχολική διοίκηση υστερεί σημαντικά σε σχέση με τις ανάγκες της εποχής. Οι υπάρχουσες λύσεις επικεντρώνονται κυρίως στη διαχείριση μαθητολογίων, απουσιών και βαθμολογιών (π.χ. το πληροφοριακό σύστημα MySchool), χωρίς να καλύπτουν την εσωτερική οργάνωση της σχολικής μονάδας ή την καθημερινή λειτουργία του Διευθυντή.

Ο Διευθυντής καλείται να διαχειριστεί κρίσιμες λειτουργίες μέσα από μεμονωμένα, ασύνδετα και πολλές φορές δυσλειτουργικά συστήματα, όπως:

- Τα πάγια υλικά και τον εξοπλισμό, μέσω εφαρμογής που είναι δυσνόητη, δύσχρηστη και χωρίς δυνατότητα αποδοτικής παρακολούθησης.
- Την καταγραφή και διαχείριση εισερχομένων/εξερχομένων εγγράφων, χωρίς σύστημα ηλεκτρονικού πρωτοκόλλου, γεγονός που οδηγεί σε ανασφάλεια, ασυνέπεια και αδυναμία ιχνηλάτησης.
- Τη διαχείριση αναπληρωτών και κάλυψη ωρών, με προσωπικούς πίνακες και χειρόγραφες σημειώσεις.
- Τον προγραμματισμό μεταφορών μαθητών (ΚΤΕΛ/ταξί), χωρίς οργανωμένο σύστημα ή ψηφιακό αρχείο.
- Την εσωτερική αξιολόγηση της σχολικής μονάδας, με διασκορπισμένα έγγραφα και χωρίς ενιαία παρακολούθηση προόδου.
- Τη διαχείριση μαθητικών λογαριασμών ΠΣΔ, μεμονωμένα και χωρίς συγκεντρωτική επισκόπηση.
- Τις παραγγελίες σχολικών βιβλίων, μέσω ανεξάρτητων πλατφορμών χωρίς καμία ενσωμάτωση με άλλες διοικητικές διεργασίες.

Αυτό το τοπίο κατακερματισμένων λύσεων, χωρίς διασύνδεση ή κοινό περιβάλλον εργασίας, προκαλεί αναποτελεσματικότητα, καθυστερήσεις και αυξημένο ψυχολογικό και λειτουργικό φόρτο στον Διευθυντή.

Υπάρχουσες λύσεις και ελλείψεις

Η ανασκόπηση των διαθέσιμων εργαλείων και συστημάτων δείχνει ότι δεν υπάρχει μέχρι σήμερα καμία ενιαία, φορητή και ευέλικτη εφαρμογή ειδικά σχεδιασμένη για τη διοικητική υποστήριξη του Διευθυντή σχολικής μονάδας.

Αν και υπάρχουν γενικές εφαρμογές (ημερολόγια, σημειωματάρια, cloud αποθήκευση, εργαλεία task management), καμία δεν προσφέρει:

- προσαρμογή στις συγκεκριμένες απαιτήσεις της σχολικής διοίκησης,
- δυνατότητα καταγραφής συμβάντων, έκδοσης εγγράφων, παρακολούθησης αδειών,
- συγκέντρωση όλων των δεδομένων σε ένα ασφαλές και προσβάσιμο περιβάλλον,
- ή τη δυνατότητα συγχρονισμού και εξαγωγής δεδομένων σε επίσημη μορφή.

Η ψηφιακή αποδιοργάνωση αυτή αποτελεί σοβαρό εμπόδιο στην ομαλή διοικητική λειτουργία του σχολείου.

Επιλογή τεχνολογιών (Flutter, Firebase, REST API)

Η επιλογή των κατάλληλων τεχνολογιών για την ανάπτυξη της εφαρμογής ήταν καθοριστική για την επιτυχία του έργου, καθώς επηρέασε άμεσα την ευχρηστία, τη φορητότητα, την ευκολία συντήρησης, την ταχύτητα υλοποίησης και τις δυνατότητες επεκτασιμότητας. Η τεχνολογική λύση έπρεπε να ικανοποιεί συγκεκριμένες προϋποθέσεις:

- Να υποστηρίζει πολυπλατφορμική ανάπτυξη (Android και iOS) με έναν ενιαίο κώδικα.
- Να επιτρέπει γρήγορη δημιουργία φιλικού και λειτουργικού περιβάλλοντος χρήστη (UI).
- Να προσφέρει δυνατότητες αποθήκευσης, αυθεντικοποίησης χρηστών και ασφαλούς διαχείρισης δεδομένων.
- Να είναι προσβάσιμη και εφικτή ως προς την καμπύλη εκμάθησης και συντήρησης, δεδομένου του πλαισίου εφαρμογής σε σχολικό περιβάλλον.
- Να μπορεί να επεκταθεί σε μελλοντικά έργα ή διασυνδέσεις με κρατικές πλατφόρμες (όπως το MySchool ή το gov.gr).

Με βάση τα παραπάνω, επιλέχθηκαν δύο βασικές τεχνολογικές κατευθύνσεις:

Flutter: Πλαίσιο ανάπτυξης εφαρμογών κινητού

Το Flutter είναι ένα framework ανοικτού κώδικα που αναπτύσσεται από τη Google. Επιτρέπει την ανάπτυξη εφαρμογών για Android, iOS, Web και Desktop με κοινή βάση κώδικα, χρησιμοποιώντας τη γλώσσα προγραμματισμού Dart.

Πλεονεκτήματα Flutter:

- Πολυπλατφορμική ανάπτυξη: μια φορά κώδικας – πολλές πλατφόρμες.
- Hot reload: άμεση εφαρμογή αλλαγών κατά την ανάπτυξη.
- Ενσωματωμένα widgets για πλήρη έλεγχο του UI χωρίς εξάρτηση από native στοιχεία.
- Καλή τεκμηρίωση και κοινότητα: διευκολύνει την εκμάθηση και την υποστήριξη.
- Ευκολία παραμετροποίησης του περιβάλλοντος χρήστη για συγκεκριμένες ανάγκες.
- Δυνατότητα δημιουργίας responsive διεπαφών, κατάλληλων για tablet ή μελλοντική web επέκταση.

Περιορισμοί:

- Αρχική εκμάθηση της Dart απαιτεί χρόνο αν δεν υπάρχει εξοικείωση.
- Ορισμένες native δυνατότητες (π.χ. Bluetooth, τηλεφωνικές λειτουργίες) χρειάζονται εξωτερικές βιβλιοθήκες ή πρόσθετο κώδικα.
- Μεγέθη εφαρμογών κάπως μεγαλύτερα από native apps, λόγω ενσωμάτωσης runtime.

Το Flutter επελέγη ως η βασική τεχνολογία ανάπτυξης της εφαρμογής, διότι προσφέρει ταχύτητα, φορητότητα και έλεγχο στο UI, χωρίς τον φόρτο του διπλού native development.

Διαχείριση δεδομένων: Firebase vs RESTful API με Βάση Δεδομένων

Η αποθήκευση, ανάκτηση και συγχρονισμός των δεδομένων της εφαρμογής απαιτεί τη χρήση backend τεχνολογίας. Εξετάστηκαν δύο βασικές προσεγγίσεις:

Firebase (Backend as a Service)

Το Firebase είναι μια πλήρως διαχειριζόμενη πλατφόρμα της Google που περιλαμβάνει:

- Realtime Database ή Cloud Firestore (NoSQL βάσεις),
- Authentication (με email, Google λογαριασμό, κ.λπ.),
- Cloud Storage (για αρχεία, έγγραφα, φωτογραφίες),
- Hosting και Analytics.

Πλεονεκτήματα:

- Άμεση ενσωμάτωση με Flutter.
- Δεν απαιτεί υποδομή server ή εγκατάσταση.

- Εύκολη ταυτοποίηση χρηστών.
- Πραγματικός χρόνος ενημέρωσης (real-time sync).

Μειονεκτήματα:

- Περιορισμένες δυνατότητες πολύπλοκων ερωτημάτων (σε σύγκριση με SQL).
- Δυσκολία στη μεταφορά δεδομένων ή εξαγωγή προς άλλες πλατφόρμες (π.χ. MySchool).
- Κόστος αυξάνεται με την κίνηση και χρήση.

RESTful API με MySQL/PostgreSQL

Η δεύτερη προσέγγιση είναι η δημιουργία RESTful API σε server (π.χ. με Node.js, Python Flask ή PHP) που διαχειρίζεται τα δεδομένα μέσω σχεσιακής βάσης (MySQL, PostgreSQL κ.ά.).

Πλεονεκτήματα:

- Δομημένα, αναλυτικά ερωτήματα (SQL).
- Ευκολότερη διασύνδεση με υπάρχουσες δημόσιες υποδομές.
- Αυξημένος έλεγχος στη ροή και ασφάλεια δεδομένων.
- Επεκτασιμότητα για σύνθετες δομές (π.χ. πίνακες παγίων, πρωτοκόλλου κ.λπ.).

Μειονεκτήματα:

- Απαιτεί υποδομή server και τεχνική συντήρηση.
- Μεγαλύτερη πολυπλοκότητα ανάπτυξης.
- Μικρότερη ταχύτητα σε push ενημερώσεις (όχι real-time).

Σύγκριση Τεχνολογιών Ανάπτυξης Εφαρμογής

Κριτήριο	Flutter	Firebase	RESTful API + SQL
Πολυπλατφορμική ανάπτυξη	(Android/iOS/Web)	—	—
Ευκολία εκμάθησης/χρήσης	απαιτεί γνώση Dart	Πολύ εύχρηστο	Μεγαλύτερη πολυπλοκότητα
Υποστήριξη UI	Εξαιρετική υποστήριξη μέσω widgets	συνεργάζεται με Flutter	μέσω διασύνδεσης
Υποστήριξη real-time ενημέρωσης		Realtime Database / Firestore	
Διαχείριση χρηστών (Authentication)		Built-in	π.χ. μέσω JWT
Επεκτασιμότητα	modular UI	Περιορισμένη σε μεγάλα σχήματα	ΝΑΙ

Διασύνδεση με άλλες πλατφόρμες	μέσω plugins	Περιορισμένη εξαγωγή/διασύνδεση	ΝΑΙ
Επεξεργασία πολύπλοκων ερωτημάτων	—	NoSQL	SQL queries
Ανάγκη για υποδομή server	ΟΧΙ	ΟΧΙ	ΝΑΙ
Καταλληλότητα για πιλοτική υλοποίηση	ΝΑΙ	ΝΑΙ	Κατάλληλο για παραγωγή

Για τις ανάγκες της πιλοτικής φάσης, επιλέχθηκε αρχικά το Firebase, λόγω της ταχύτητας υλοποίησης και της άμεσης ενσωμάτωσης με Flutter. Το Firebase καλύπτει με ικανοποιητικό τρόπο:

- την πιστοποίηση χρηστών,
- την αποθήκευση εγγράφων και συμβάντων,
- την πραγματική χρονική ενημέρωση σε υπενθυμίσεις και ημερολόγιο.

Παράλληλα, η αρχιτεκτονική της εφαρμογής σχεδιάστηκε ώστε να είναι modular, και να μπορεί μελλοντικά να συνδεθεί με RESTful API και σχεσιακή βάση δεδομένων, εφόσον απαιτηθεί αυξημένη παραμετροποίηση, ανάλυση ή διαλειτουργικότητα με δημόσιες βάσεις δεδομένων.

Η τεχνολογική επιλογή βασίστηκε σε κριτήρια απλότητας, φορητότητας, επεκτασιμότητας και πρακτικής υλοποίησης. Η απόφαση να χρησιμοποιηθεί το Flutter ως εργαλείο ανάπτυξης, σε συνδυασμό με Firebase στην αρχική φάση, επιτρέπει την ταχεία δημιουργία μιας λειτουργικής, φορητής εφαρμογής, προσαρμοσμένης στις ανάγκες ενός Διευθυντή σχολικής μονάδας.

Ανάλυση Απαιτήσεων

Η ανάλυση απαιτήσεων είναι κρίσιμο στάδιο στη διαδικασία ανάπτυξης λογισμικού. Αποσκοπεί στη συστηματική κατανόηση του τι αναμένεται να υλοποιηθεί, ποιοι είναι οι χρήστες του συστήματος, ποιες λειτουργίες πρέπει να προσφέρει και ποια χαρακτηριστικά ποιότητας πρέπει να πληροί.

Στο παρόν κεφάλαιο αναλύονται σε βάθος οι απαιτήσεις του συστήματος, όπως αυτές προέκυψαν από την καταγραφή αναγκών που παρουσιάστηκε στο Κεφάλαιο 3.

Συλλογή και κατηγοριοποίηση αναγκών

Η συλλογή των απαιτήσεων βασίστηκε:

- σε **προσωπική εμπειρία** Διευθυντή,
- στην **παρατήρηση της λειτουργίας** σχολικής μονάδας,
- και σε **συζητήσεις με εκπαιδευτικούς και στελέχη** της διοίκησης.

Οι ανάγκες κατανεμήθηκαν σε δύο βασικές κατηγορίες:

- **Λειτουργικές απαιτήσεις**: Περιγράφουν τι ακριβώς πρέπει να κάνει το σύστημα.
- **Μη λειτουργικές απαιτήσεις**: Αφορούν χαρακτηριστικά ποιότητας του συστήματος, όπως ασφάλεια, απόδοση και χρηστικότητα

Χρήστες και ρόλοι

Αν και η εφαρμογή σχεδιάστηκε κυρίως για τον Διευθυντή σχολικής μονάδας, προβλέπεται και η πιθανή μελλοντική υποστήριξη πρόσβασης για άλλους ρόλους.

1. **Βασικός Χρήστης – Διευθυντής Γυμνασίου**
    1. Έχει πλήρη πρόσβαση και δικαιώματα διαχείρισης.
    2. Εγγράφεται μέσω email ή Google λογαριασμού.
    3. Εκτελεί όλες τις λειτουργίες της εφαρμογής (καταγραφή, δημιουργία, οργάνωση, αναζήτηση).

2. **Πιθανές Επεκτάσεις – Μελλοντικοί Ρόλοι**
    1. **Υποδιευθυντής:** περιορισμένη πρόσβαση σε επιμέρους λειτουργίες (π.χ. διαχείριση αδειών).
    2. **Εκπαιδευτικοί:** για υποβολή αιτήσεων αδειών ή παρακολούθηση ανακοίνωσης εγγράφων.
    3. **Γονείς**: για ειδοποιήσεις (push) και λήψη προσωπικών ενημερώσεων (π.χ. ραντεβού, συμπεριφορές).

Για την πιλοτική έκδοση, υλοποιείται μόνο ο ρόλος του Διευθυντή (μονοχρηστικό σενάριο χρήσης).

Λειτουργικές απαιτήσεις

Οι λειτουργικές απαιτήσεις εξειδικεύουν με σαφήνεια τις δυνατότητες που πρέπει να παρέχει η εφαρμογή στον χρήστη της. Οι βασικές ενότητες περιγράφονται ως εξής:

1. **Διαχείριση Ραντεβού**
   1.1. Καταχώριση ραντεβού με στοιχεία (τίτλος, πρόσωπο, λόγος, ημερομηνία/ώρα).
   1.2. Προβολή ημερολογίου με φίλτρα ανά ημέρα/εβδομάδα.
   1.3. Ειδοποιήσεις για επερχόμενα ραντεβού.
   1.4. Καταγραφή σύντομων σημειώσεων ανά ραντεβού.
2. **Δημιουργία και Διαχείριση Εγγράφων**
   2.1. Δυνατότητα δημιουργίας εγγράφου από πρότυπο.
   2.2. Πεδία που συμπληρώνονται αυτόματα (ημερομηνία, ονοματεπώνυμο, θέμα).
   2.3. Επεξεργασία, αποθήκευση και εξαγωγή εγγράφου (π.χ. σε PDF).
   2.4. Ταξινόμηση σε κατηγορίες (π.χ. αποφάσεις, ειδοποιήσεις, διοικητικά).
   2.5. Αναζήτηση με λέξεις-κλειδιά ή ημερομηνία.
3. **Καταγραφή Συμβάντων / Ημερολόγιο**
   3.1. Εισαγωγή συμβάντος με κατηγοριοποίηση.
   3.2. Καταχώριση ημερομηνίας, περιγραφής, σχετικών προσώπων.
   3.3. Επισύναψη αρχείων ή φωτογραφιών.
   3.4. Προβολή ανά ημερολογιακή περίοδο.
4. **Ταυτοποίηση και Ασφάλεια Πρόσβασης**
   4.1. Είσοδος με email/κωδικό ή χρήση Google λογαριασμού.
   4.2. Διασφάλιση πρόσβασης μόνο στον εξουσιοδοτημένο χρήστη.
   4.3. Μελλοντική υποστήριξη για 2FA (two-factor authentication).

Μη λειτουργικές απαιτήσεις

Οι μη λειτουργικές απαιτήσεις αφορούν την ποιότητα και τη συμπεριφορά της εφαρμογής ανεξαρτήτως λειτουργικότητας:

Κατηγορία	**Απαίτηση**
Φορητότητα	Συμβατότητα με Android και iOS μέσω Flutter.
Απόδοση	Άμεση απόκριση σε ενέργειες (<300ms), ελαχιστοποίηση καθυστερήσεων φόρτωσης.
Ασφάλεια	Κρυπτογράφηση δεδομένων, πιστοποίηση χρήστη, προστασία από μη εξουσιοδοτημένη πρόσβαση.
Ευχρηστία	Εργονομική διεπαφή, σαφές μενού, συμβολική πλοήγηση, προσαρμογή για κινητό.

Κατηγορία	Απαίτηση
Διαθεσιμότητα	Δυνατότητα χρήσης εκτός σχολικού δικτύου, 24/7 λειτουργία.
Συντηρησιμότητα	Καθαρός, modular κώδικας. Υποστήριξη ενημερώσεων.
Επεκτασιμότητα	Δυνατότητα προσθήκης λειτουργιών: πολλαπλοί χρήστες, προγράμματα, ειδοποιήσεις.
Συγχρονισμός	Realtime (Firebase) ή περιοδικός (REST API) συγχρονισμός με cloud.
Δημιουργία Αντιγράφων	Backup δεδομένων τοπικά ή στο cloud.
Συμβατότητα	Υποστήριξη μελλοντικής διασύνδεσης με MySchool, gov.gr ή άλλα συστήματα.

Σχεδίαση της Εφαρμογής

Η σχεδίαση της εφαρμογής αποτελεί το κρίσιμο ενδιάμεσο στάδιο μεταξύ της ανάλυσης απαιτήσεων και της υλοποίησης. Αποσκοπεί στο να δώσει σαφή μορφή στην αρχιτεκτονική, τις οντότητες, τις λειτουργικές ροές και τη διεπαφή χρήστη. Η παρούσα εφαρμογή σχεδιάστηκε ώστε να είναι απλή, φορητή, εργονομική και πλήρως προσαρμοσμένη στο ρόλο και τις ανάγκες του Διευθυντή Γυμνασίου.

Αρχιτεκτονική συστήματος

Η εφαρμογή σχεδιάστηκε με βάση τη λογική της **client-server αρχιτεκτονικής**, στην οποία το κινητό τηλέφωνο του χρήστη (client) επικοινωνεί με ένα **backend σύστημα αποθήκευσης και διαχείρισης δεδομένων**.

Επιμέρους συνιστώσες:

- **Frontend (Flutter)**: Το περιβάλλον του χρήστη, κατασκευασμένο εξ ολοκλήρου με το framework Flutter, ώστε να εξασφαλίζεται κοινή βάση κώδικα για Android και iOS, απόκριση, φορητότητα και σύγχρονη σχεδίαση.
- **Backend (Firebase ή REST API)**: Διαχείριση χρηστών, αποθήκευση εγγράφων, καταγραφή ενεργειών και συγχρονισμός με την εφαρμογή. Η επιλογή του backend αναλύεται στο Κεφάλαιο 4, με προτίμηση στο Firebase για λόγους ευχρηστίας και απλότητας.
- **Βάση δεδομένων**: Ανάλογα με την επιλογή backend, χρησιμοποιείται είτε η NoSQL βάση του Firebase (Firestore), είτε σχεσιακή βάση (MySQL/PostgreSQL) με RESTful API.

Οντότητες του Συστήματος

Οι βασικές οντότητες του συστήματος και οι μεταξύ τους σχέσεις είναι οι εξής:

Οντότητα	Πεδία / Ιδιότητες
Χρήστης	ID, Όνομα, Email, Ρόλος, Κωδικός
Ραντεβού	Τίτλος, Ημερομηνία/Ώρα, Πρόσωπο, Σχόλια
Άδεια	Εκπαιδευτικός, Τύπος, Ημερομηνίες
Έγγραφο	Τίτλος, Κατηγορία, Περιεχόμενο
Συμβάν	Τύπος, Περιγραφή, Ημερομηνία, Συσχετιζόμενα άτομα
Αρχείο	Όνομα, Σύνδεσμος αποθήκευσης (Cloud), Συνδεδεμένο αντικείμενο

Διεπαφή Χρήστη (User Interface)

Η διεπαφή χρήστη σχεδιάστηκε με γνώμονα την εργονομία και την ταχύτητα πρόσβασης στις βασικές λειτουργίες. Τα βασικά στοιχεία περιλαμβάνουν:

- **Αρχική οθόνη (Dashboard)**: με συντομεύσεις για τα κύρια υποσυστήματα (Ραντεβού, Άδειες, Έγγραφα, Ημερολόγιο, Αναζήτηση).
- **Πλοήγηση μέσω menu ή tabs**, με λογική κατηγοριοποίηση ανά λειτουργικό πεδίο.
- **Απλός και καθαρός σχεδιασμός**, με χρήση χρωμάτων για διακριτές κατηγορίες και κατάσταση
- **Ειδοποιήσεις (push notifications)** για υπενθυμίσεις, νέα συμβάντα ή ανανεωμένες καταχωρίσεις.
- **Συμβατότητα με μικρές οθόνες** και προσαρμογή διεπαφής για χρήση στο κινητό τηλέφωνο με το ένα χέρι.

Προδιαγραφές Ασφάλειας και Δικαιωμάτων Πρόσβασης

Η εφαρμογή σχεδιάστηκε ως μονοχρηστική για την πιλοτική φάση, αλλά προβλέπεται επεκτασιμότητα σε πολλαπλούς ρόλους με διαφορετικά επίπεδα δικαιωμάτων:

- **Διευθυντής**: πλήρης πρόσβαση και διαχείριση όλων των λειτουργιών.
- **Υποδιευθυντής**: δυνατότητα παρακολούθησης ή καταχώρισης χωρίς ορισμένα δικαιώματα έγκρισης.
- **Εκπαιδευτικός**: πρόσβαση για αιτήσεις αδειών ή προβολή προσωπικών ραντεβού.
- **Γονέας**: περιορισμένη προβολή ειδοποιήσεων ή ραντεβού.

Όλα τα δεδομένα αποθηκεύονται με κρυπτογράφηση και η είσοδος γίνεται με authentication (Google ή email/κωδικός). Υποστηρίζεται η υλοποίηση 2FA σε επόμενο στάδιο.

Υλοποίηση – Αρχιτεκτονική Εφαρμογής - Εφαρμογή

Περιβάλλον ανάπτυξης (Flutter, Firebase, Google Scripts, Proxy Server)

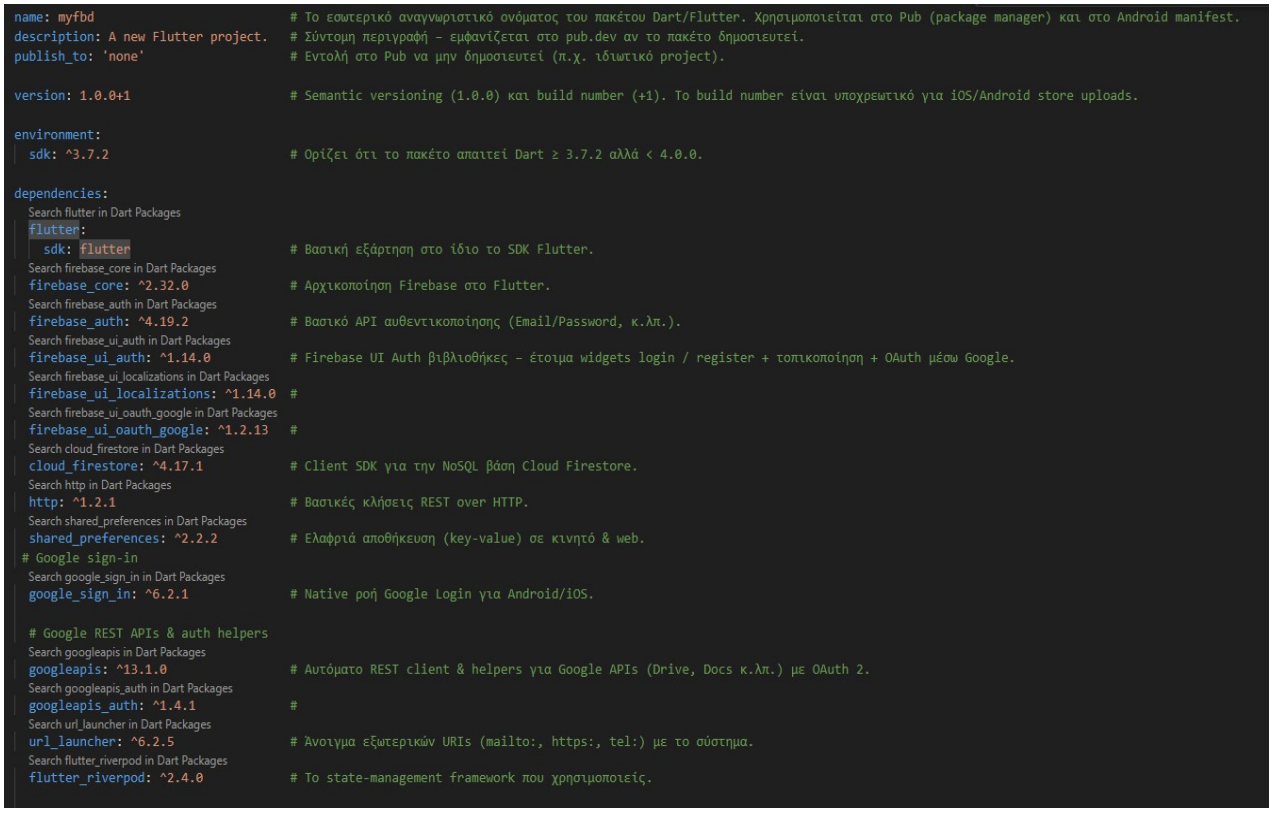
Στο κεφάλαιο αυτό περιγράφουμε τις τεχνολογίες και τα εργαλεία που αξιοποιήθηκαν για την ανάπτυξη της εφαρμογής Flutter. Για κάθε τεχνολογία παραθέτουμε σύντομη θεωρητική εισαγωγή και ενδεικτικά αποσπάσματα κώδικα από το αποθετήριο του έργου.

Flutter

Flutter είναι ένα open-source UI SDK της Google που επιτρέπει την ανάπτυξη εγγενών εφαρμογών (native apps) για Android, iOS, web και desktop από κοινή βάση κώδικα σε Dart. Τα βασικά πλεονεκτήματά του για το έργο μας ήταν:

1. Hot reload για άμεση προεπισκόπηση αλλαγών.
2. Πλούσια συλλογή από widgets (Material & Cupertino) που εξασφάλισαν συνεπές UI/UX.
3. Υψηλή απόδοση χάρη στη μηχανή Skia.

Στο αρχείο pubspec.yaml δηλώθηκαν οι εξαρτήσεις και οι βιβλιοθήκες που απαιτήθηκαν στην ανάπτυξη της εφαρμογής παρακάτω βλέπουμε ένα τμήμα του.



FireBase

Το Firebase είναι μια ολοκληρωμένη πλατφόρμα backend της Google για την ανάπτυξη web και mobile εφαρμογών. Για την παρούσα εφαρμογή Flutter χρησιμοποιήθηκαν οι ακόλουθες υπηρεσίες του Firebase:

- Firebase Authentication για αυθεντικοποίηση χρηστών.
- Cloud Firestore ως βάση δεδομένων NoSQL σε πραγματικό χρόνο.
- Firebase Hosting / Cloud Functions για τη φιλοξενία Web Apps και υποστήριξη server-side λειτουργιών (μέσω Google Apps Script και proxy endpoints).

Cloud Firestore: NoSQL Βάση Δεδομένων

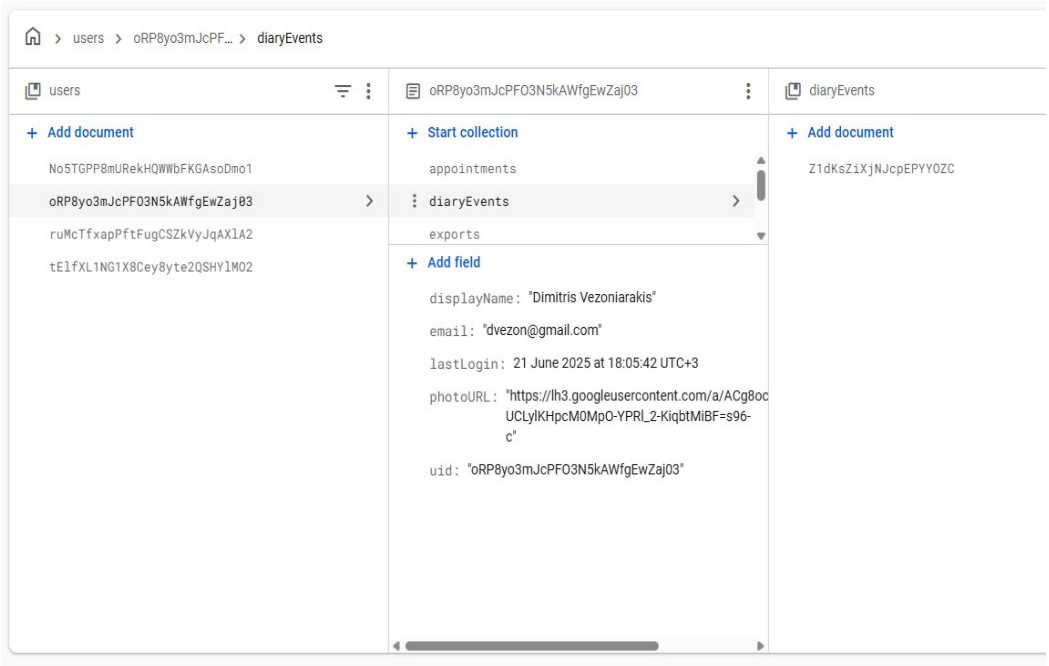
Η Cloud Firestore είναι μια scalable βάση δεδομένων εγγράφων (document database), σχεδιασμένη για συγχρονισμό δεδομένων σε πραγματικό χρόνο. Τα βασικά χαρακτηριστικά της περιλαμβάνουν:

- Δομή με Συλλογές και Έγγραφα (Collections & Documents).
- Υποστήριξη offline αποθήκευσης.
- Κανόνες ασφαλείας (security rules) για περιορισμό πρόσβασης.
- Ενσωμάτωση με Firebase Authentication για ανά χρήστη δεδομένα.

Δομή Δεδομένων της Εφαρμογής:

```
users (collection)
  └── {userId} (document)
        ├── appointments (subcollection)
        ├── diaryEvents (subcollection)
        └── mydocs (subcollection)
```

Δείγμα από τη console.firebase.google.com της Βάσης Δεδομένων της εφαρμογής



Κανόνες (Rules) της Βάσης Δεδομένων.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {

    // root-doc του χρήστη
    match /users/{userId} {
      allow read, write: if request.auth != null
                         && request.auth.uid == userId;
    }

    // κάθε υποσυλλογή του χρήστη (1 επίπεδο βάθος)
    match /users/{userId}/{collection}/{doc} {
      allow read, write: if request.auth != null
                         && request.auth.uid == userId;
    }

    // optional: read-only global fallback
    match /settings/global/app {
      allow read: if true;
    }
  }
}

```

Google Apps Script

Το Google Apps Script είναι μια πλατφόρμα ανάπτυξης που βασίζεται σε JavaScript και επιτρέπει την αυτοματοποίηση και προσαρμογή των υπηρεσιών Google, όπως τα Google Docs, Sheets και Drive. Στο πλαίσιο της εφαρμογής, χρησιμοποιήθηκε για τη δυναμική δημιουργία εγγράφων Google Docs βάσει προκαθορισμένων προτύπων και την εξαγωγή δεδομένων σε αρχεία τύπου Google Sheets.

Λειτουργίες που Υλοποιήθηκαν με Google Apps Script

1. Δημιουργία προσωποποιημένων εγγράφων με βάση template.
2. Καταγραφή εγγράφων σε Google Sheet για ιστορικό και αναζήτηση.
3. Δημοσίευση ως Web App για πρόσβαση από Flutter μέσω POST.

Πλεονεκτήματα της Χρήσης Google Apps Script

1. Δεν απαιτείται dedicated server.
2. Άμεση διασύνδεση με τις υπηρεσίες Google.
3. Ευέλικτη δημιουργία εγγράφων από mobile περιβάλλον.
4. Δωρεάν χρήση με τον Google λογαριασμό του χρήστη.

Proxy μέσω Firebase Functions

Κατά την ανάπτυξη της εφαρμογής, παρατηρήθηκε ότι η απευθείας αποστολή HTTP POST αιτημάτων από το Flutter προς το Google Apps Script Web App οδηγούσε σε σφάλματα CORS (Cross-Origin Resource Sharing). Τα Web Apps της Google, όταν είναι δημόσια, απορρίπτουν αιτήματα που προέρχονται από άλλους origins (όπως από κινητές εφαρμογές ή από το localhost).

Για την επίλυση αυτού του περιορισμού, υλοποιήσαμε ένα ενδιάμεσο proxy server με χρήση Firebase Cloud Functions, που λειτουργεί ως ενδιάμεσος μεταφορέας (server-to-server):

- Δέχεται το αίτημα από την εφαρμογή Flutter χωρίς CORS περιορισμούς.
- Το προωθεί διαφανώς προς το Web App του Google Apps Script.
- Επιστρέφει την απόκριση πίσω στην εφαρμογή.

Η λειτουργία αυτή προσφέρει επιπλέον ασφάλεια, καθώς αποκρύπτει τη διεύθυνση του Google Script από τον τελικό χρήστη και επιτρέπει logging ή ελέγχους αυθεντικοποίησης στο μέλλον.

Αυτή η αρχιτεκτονική επέτρεψε την αξιόπιστη, ασφαλή και χωρίς περιορισμούς επικοινωνία της Flutter εφαρμογής με τις υπηρεσίες Google (Docs/Sheets) μέσω server-side logic. Ακολουθεί ένα τμήμα της.

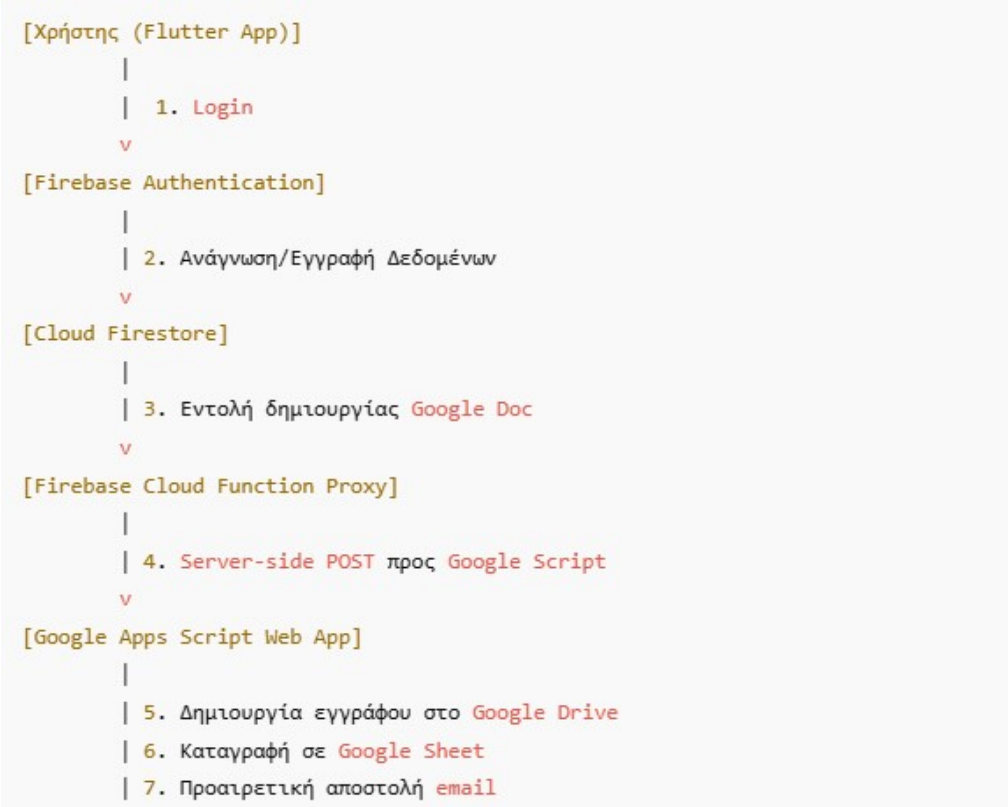
```
15  export const proxyExport = onRequest(
16    {
17      region: "us-central1",
18      timeoutSeconds: 60,
19      memory: "256MiB",
20      cors: true, // Αυτόματα CORS για *
21    },
22    async (req, res) => {
23      if (req.method === "OPTIONS") {
24        // CORS pre-flight handled by 'cors: true' αλλά κρατάμε fallback
25        return res.status(204).send("");
26      }
27
28      const body = req.body || {};
29      const { scriptUrl } = body;
30
31      if (!scriptUrl) {
32        return res.status(400).send("Missing scriptUrl in body");
33      }
34
35      // --- Host έλεγχος ----------------------------------------------------
36      try {
37        const host = new URL(scriptUrl).host;
38        const allowed = ALLOWED_HOSTS.some((h) => host.endsWith(h));
39        if (!allowed) {
40          logger.warn("Host not allowed:", host);
41          return res.status(400).send("Host not allowed");
42        }
43      } catch (err) {
44        return res.status(400).send("Invalid scriptUrl");
45      }
```

Ενοποίηση Τεχνολογιών – Αρχιτεκτονική της Εφαρμογής

Η εφαρμογή βασίστηκε σε αρχιτεκτονική client-server με υποστήριξη από cloud υπηρεσίες της Google (Firebase & Google Workspace). Το περιβάλλον ανάπτυξης περιλάμβανε:

1. Flutter (Client)
2. Firebase (Authentication, Firestore, Cloud Functions)
3. Google Apps Script (Docs, Sheets, Drive automation)

Η επικοινωνία μεταξύ των επιμέρους συστημάτων ακολουθεί την παρακάτω λογική:



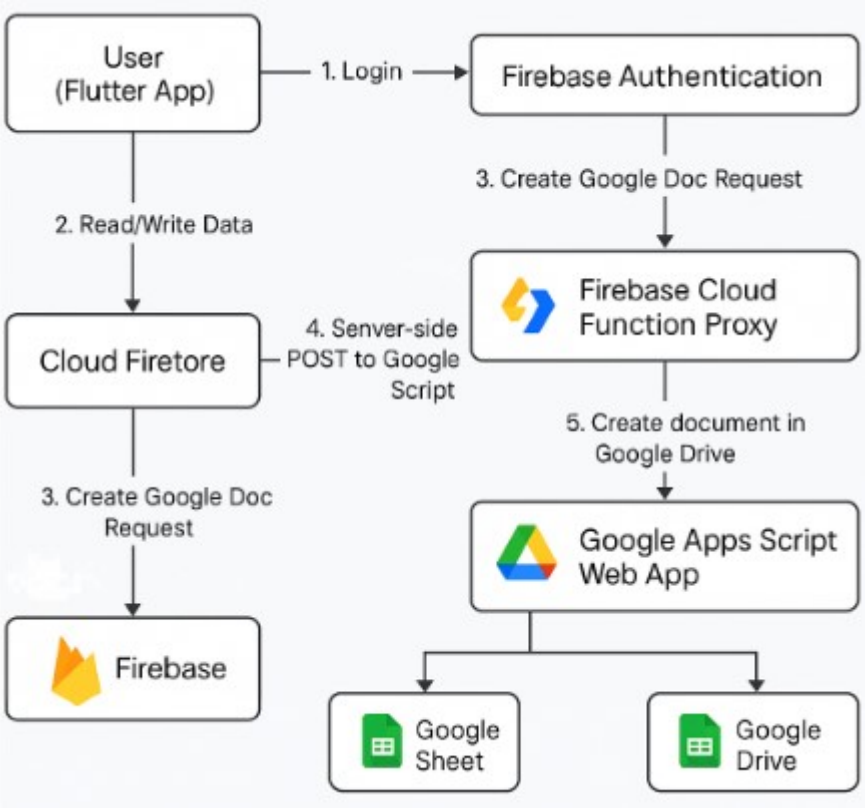
Περιγραφή Ροής:

1. Ο χρήστης συνδέεται μέσω Firebase Authentication (με email ή Google Sign-In).
2. Τα δεδομένα του αποθηκεύονται στη βάση Cloud Firestore, οργανωμένα ανά uid.
3. Κατά τη δημιουργία εγγράφου, η εφαρμογή στέλνει POST αίτημα στο proxy (Cloud Function).
4. Η Cloud Function αναλαμβάνει να προωθήσει το αίτημα στο Google Apps Script Web App.
5. Το Apps Script δημιουργεί νέο αρχείο Google Docs με βάση το επιλεγμένο πρότυπο.
6. Καταγράφει το νέο αρχείο σε Google Sheet για ιστορικό και παρακολούθηση.
7. Τέλος, προαιρετικά, αποστέλλει email ειδοποίησης σε διαχειριστή ή χρήστη.

Πλεονεκτήματα της Αρχιτεκτονικής:

- Υψηλή επεκτασιμότητα και ευελιξία
- Δεν απαιτείται δικός μας server (όλα τα endpoints είναι serverless)
- Εύκολη ενσωμάτωση Google υπηρεσιών μέσω Apps Script
- Αντιμετώπιση θεμάτων CORS μέσω του proxy
- Ασφαλής πρόσβαση βάσει Firebase Auth UID

Διάγραμμα της αρχιτεκτονικής:



Διαχείριση ραντεβού και υπενθυμίσεων

Η οθόνη «Διαχείριση ραντεβού» υλοποιεί ένα πλήρες υποσύστημα προγραμματισμού συναντήσεων: αφού ανακτήσει το μοναδικό uid του τρέχοντος χρήστη μέσω Firebase Auth, συνδέεται δυναμικά στο υπο-collection users/{uid}/appointments του Cloud Firestore και, με ένα StreamBuilder, αντλεί σε πραγματικό χρόνο τα επόμενα έως τριάντα ραντεβού ταξινομημένα κατά ημερομηνία· έτσι κάθε εισαγωγή, τροποποίηση ή διαγραφή ενημερώνει αυτόματα το UI χωρίς ανανέωση.

Η κεντρική λίστα, αγκαλιασμένη σε προσαρμοσμένο BorderedBox, παρουσιάζει τίτλο, σημειώσεις και χρονικό στίγμα κάθε εγγραφής, ενώ δύο εικονίδια ενεργειών (✎ και 🗑) επιτρέπουν αντίστοιχα άμεση επεξεργασία ή οριστική διαγραφή με μία κλήση στην doc.reference.update() ή delete().

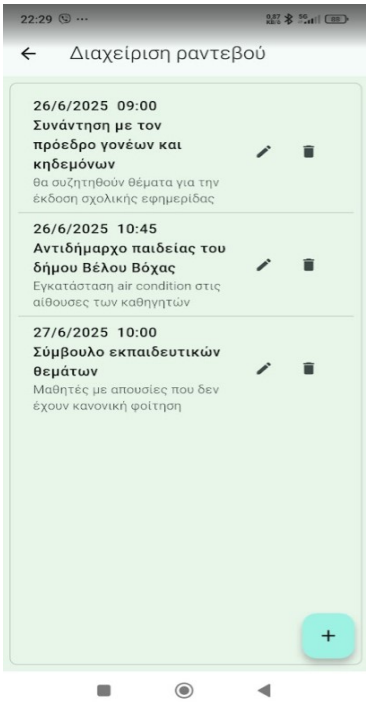
Η προσθήκη/τροποποίηση ενεργοποιεί έναν διάλογο τύπου Material (AlertDialog) που φιλοξενεί πεδία τίτλου και παρατηρήσεων, καθώς και κουμπί επιλογής ημερομηνίας/ώρας· το κουμπί εκκινεί βαθμιδωτά τα showDatePicker και showTimePicker, επιβάλλοντας όριο από τη σημερινή ημερομηνία έως πέντε έτη μετά, ενώ η τελική επιλογή απεικονίζεται αθροιστικά στο ίδιο κουμπί για άμεση οπτική επιβεβαίωση.

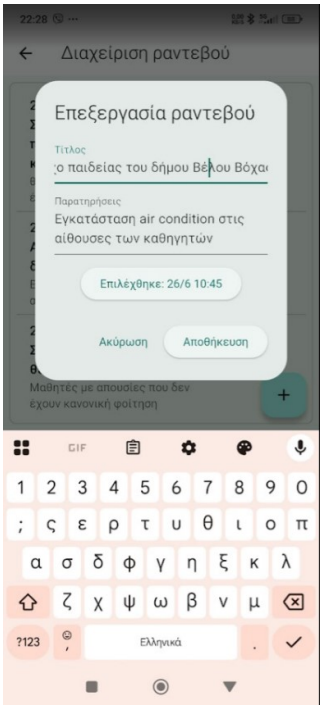
Κατά την αποθήκευση, το σύστημα ελέγχει υποχρεωτικά την ύπαρξη τίτλου και χρονικού στίγματος εμφανίζοντας προειδοποιητικό SnackBar σε περίπτωση παράλειψης· αλλιώς, δημιουργεί νέο document με πεδία title, notes, dateTime και automatized createdAt (ή ενημερώνει τα αντίστοιχα πεδία και updatedAt σε υπάρχον), μετατρέποντας το επιλεγμένο DateTime σε Timestamp για συνέπεια με την αποθήκευση Firestore. Ταυτόχρονα, όλες οι λειτουργίες κληρονομούν τα πλεονεκτήματα offline-first του Firestore (τοπική cache, μετέπειτα συγχρονισμός), ενώ το συνεπές Material UI εξασφαλίζει διαισθητική χρήση: από το πλωτό κουμπί «+» για δημιουργία, έως το ανανεωμένο πρασινωπό καρτελάκι που επιβεβαιώνει μια επιτυχημένη τροποποίηση στη λίστα.Με αυτόν τον τρόπο, η ενότητα προσφέρει στον χρήστη έναν εύχρηστο, ασφαλή και πλήρως real-time μηχανισμό διαχείρισης ραντεβού, παραμένοντας ταυτόχρονα επεκτάσιμη για λειτουργίες όπως push-ειδοποιήσεις, συγχρονισμό με Google Calendar ή φίλτρα προβολής ιστορικών συναντήσεων.

Ροή επεξεργασίας ραντεβού:

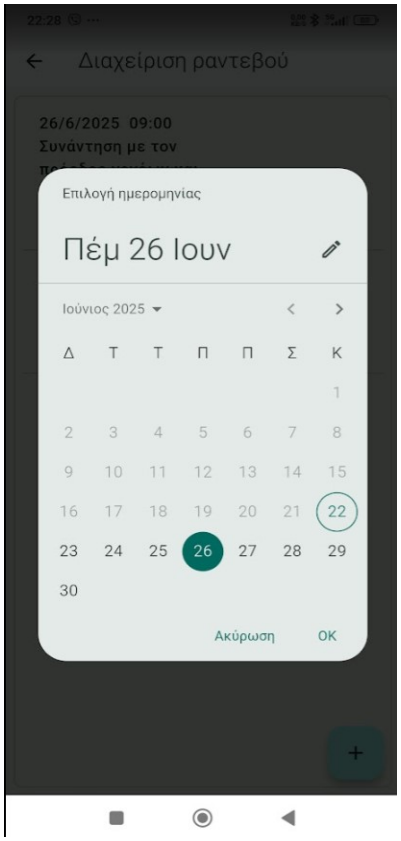
Η παρακάτω αλληλουχία εικόνων παρουσιάζει τη ροή:

Ο χρήστης πατάει ✎ σε ένα υπάρχον ραντεβού ή (+) και ανοίγει το διάλογο επεξεργασίας





Επιλέγει νέα ημερομηνία από το ημερολόγιο την ώρα σε ρολόι 24ωρης μορφής.



Η διαδικασία είναι διαισθητική και συμμορφώνεται με το Material Design, διευκολύνοντας τον χρήστη να διαχειρίζεται τα ραντεβού του γρήγορα και με ακρίβεια.

Υλοποίηση

Ο κώδικας της υλοποίησης γίνεται στο αρχείο appointments_page.dart, το αρχείο ορίζεται ως StatelessWidget, δημιουργεί δυναμικό Stream προς τη συλλογή users/{uid}/appointments του Firestore, παρακολουθεί real-time αλλαγές και αποδίδει τη λίστα ραντεβού, ενώ μέσω καθαρά ορισμένων helper μεθόδων (_appointmentsRef, _showAppointmentDialog) επιτυγχάνει διαχωρισμό ευθυνών· η διάδραση με τον χρήστη γίνεται αποκλειστικά με Material components (ListTile, AlertDialog, Date/Time pickers), διατηρώντας τοπικό state μόνο στο διάλογο, και κάθε ενέργεια (Create/Update/Delete) μεταφράζεται σε μία μοναδική Firestore κλήση, εξασφαλίζοντας απλότητα, συνοχή και αυτοματοποιημένο συγχρονισμό του UI. Στο Παράρτημα (Ε) Διαχείριση Ραντεβού) παρουσιάζεται λεπτομερής περιγραφή περιγραφή. Η παρακάτω εικόνα δείχνει ένα μέρος του κώδικα.

```dart
import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'mylib/mywidgets.dart';

// ---------------- Firestore ----------------
CollectionReference<Map<String, dynamic>> _appointmentsRef() {
  final uid = FirebaseAuth.instance.currentUser!.uid;
  return FirebaseFirestore.instance
      .collection('users')
      .doc(uid)
      .collection('appointments');
}

class AppointmentsPage extends StatelessWidget {
  const AppointmentsPage({super.key});

  @override
  Widget build(BuildContext context) {
    final now = Timestamp.now();

    return Scaffold(
      appBar: AppBar(title: const Text('Διαχείριση ραντεβού')),
      body: StreamBuilder<QuerySnapshot<Map<String, dynamic>>>(
        stream:
            _appointmentsRef()
                .orderBy('dateTime')
                .where('dateTime', isGreaterThan: now)
                .limit(30)
                .snapshots(),
        builder: (context, snap) {
          if (snap.connectionState == ConnectionState.waiting) {
            return const Center(child: CircularProgressIndicator());
          }
          if (snap.hasError) {
            return Center(child: Text('Σφάλμα: ${snap.error}'));
          }

          final docs = snap.data!.docs;
          if (docs.isEmpty) {
```

Διαχείριση αδειών και εγγράφων

Η οθόνη «Έγγραφα» που υλοποιείται μέσω της κλάσης TemplateDropdownPage στο αρχείο create_doc.dart παρέχει στον χρήστη μια πλήρως διαδραστική και δυναμικά προσαρμόσιμη εμπειρία δημιουργίας, διαχείρισης και εξαγωγής υπηρεσιακών εγγράφων μέσα από ένα φιλικό και ευέλικτο περιβάλλον.

Η λειτουργικότητα της σελίδας εστιάζει στην υποστήριξη διαφόρων τύπων εγγράφων, τα οποία ανακτώνται δυναμικά κατά την εκτέλεση της εφαρμογής από τον φάκελο προτύπων στο Google Drive ή από τοπικά μεταδεδομένα στο Firestore. Τέτοιοι τύποι περιλαμβάνουν ενδεικτικά τις «Άδεια Κανονική», «Υπερωρία», ή «Αίτηση Μετακίνησης», αλλά η εφαρμογή έχει σχεδιαστεί ώστε να μην εξαρτάται από στατικά δηλωμένες τιμές. Οποιοδήποτε νέο πρότυπο αρχείο προστεθεί στον φάκελο του Drive μπορεί να ενσωματωθεί αυτόματα στη λειτουργία της εφαρμογής, καθώς κατά το export λαμβάνεται υπόψη μόνο το templateId και τα πεδία που ορίζει ρητά αυτό το πρότυπο, χωρίς καμία απαίτηση για αλλαγή στον πηγαίο κώδικα του Flutter.

Ο χρήστης ξεκινά επιλέγοντας τον τύπο εγγράφου από ένα DropdownButtonFormField, ενεργοποιώντας έτσι ένα StreamBuilder που παρακολουθεί σε πραγματικό χρόνο τα αποθηκευμένα έγγραφα του αντίστοιχου τύπου στη διαδρομή users/{uid}/templates/{templateId}/docs. Κάθε υπάρχουσα εγγραφή εμφανίζεται ως κάρτα με θέμα, ημερομηνία, στοιχεία αποστολής και δυνατότητες για άμεση επεξεργασία (✎), διαγραφή (🗑) ή εξαγωγή εγγράφου σε Google Docs (☁), χρησιμοποιώντας τις μεθόδους του drive_service.dart και μια Firebase Web App Script ή Cloud Function.

Πατώντας το πλωτό κουμπί «+», εμφανίζεται φόρμα εισαγωγής/επεξεργασίας, η οποία περιλαμβάνει πεδία όπως: όνομα υπαλλήλου, φύλο, κλάδος, ημερομηνία έναρξης/λήξης, αριθμός ημερών, αποδέκτες, και επιπλέον μεταδεδομένα όπως eidikotita, protocol, filename. Οι ημερομηνίες επιλέγονται μέσω showDatePicker, ενώ η επεξεργασία των πεδίων συνοδεύεται από κατάλληλο validation με χρήση GlobalKey<FormState> ώστε να διασφαλίζεται η εγκυρότητα και πληρότητα της υποβαλλόμενης πληροφορίας.

Η υποβολή της φόρμας αποθηκεύει τα δεδομένα στο Firestore και συγχρονίζει αμέσως την κατάσταση UI με την αποθηκευμένη πληροφορία, μέσω του stream. Επιπλέον, προτού εξαχθεί το έγγραφο, ο χρήστης έχει τη δυνατότητα να προβάλει πλήρως όλα τα πεδία σε ένα modal AlertDialog με key-value εμφάνιση, εξασφαλίζοντας οπτική επιβεβαίωση για ακρίβεια στο περιεχόμενο.

Η εξαγωγή εγγράφων υλοποιείται μέσω POST αιτήματος προς Web App Script της Google ή μέσω callable Cloud Function, η οποία αξιοποιεί template εγγράφου από Google Drive, τοποθετεί τα δυναμικά δεδομένα και δημιουργεί προσωποποιημένο Google Doc, με καταγραφή σε Firestore και αποθήκευση σε ειδικό φάκελο στο Drive.

Η αρχιτεκτονική διαχωρίζει πλήρως τα concerns: η κλάση χειρίζεται το UI και την αλληλεπίδραση, ενώ οι κλήσεις σε Firestore, Drive ή Google APIs είναι απομονωμένες σε αρχεία όπως drive_service.dart και myfunctions.dart. Αυτός ο διαχωρισμός προσφέρει συντηρησιμότητα και επεκτασιμότητα, επιτρέποντας την προσθήκη νέων τύπων

εγγράφων και αλλαγές στη δομή δεδομένων χωρίς ανάγκη για ουσιαστικές τροποποιήσεις στο βασικό κώδικα της εφαρμογής.

Τέλος, η συνολική σχεδίαση υιοθετεί πλήρως τις αρχές του Material Design και εξασφαλίζει προσβασιμότητα, συνέπεια και φιλικότητα προς τον τελικό χρήστη, συνδυάζοντας αποτελεσματικά λειτουργικότητα backend και ευχρηστία frontend.

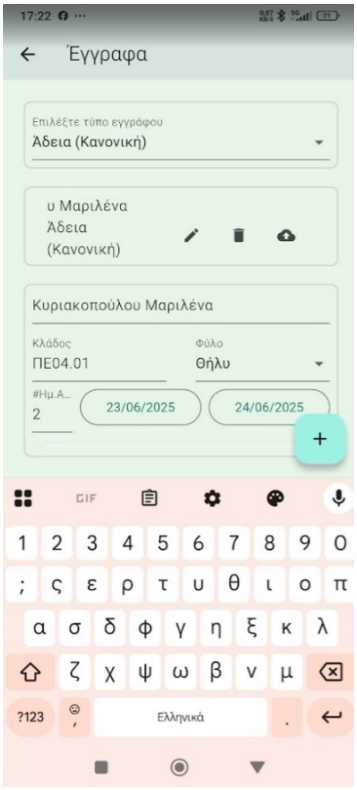
Ροή επεξεργασίας εγγράφου

Η ροή επεξεργασίας εγγράφου, όπως παρουσιάζεται στην οθόνη create_doc.dart, είναι σχεδιασμένη με στόχο την πλήρη ευελιξία στη δημιουργία, διαχείριση και εξαγωγή εγγράφων με βάση δυναμικά templates. Ο χρήστης αρχικά καλείται να επιλέξει τύπο εγγράφου από μια λίστα που δημιουργείται δυναμικά κατά την εκτέλεση, διαβάζοντας τα templates από φάκελο στο Google Drive. Αυτή η αρχιτεκτονική επιτρέπει στην εφαρμογή να προσαρμόζεται αυτόματα σε νέους τύπους εγγράφων χωρίς να απαιτείται καμία τροποποίηση στον πηγαίο κώδικα. Αμέσως μετά την επιλογή προτύπου, εμφανίζονται τα σχετικά αποθηκευμένα έγγραφα του χρήστη από τη Firestore, οργανωμένα σε κάρτες, με επιλογές για επεξεργασία, διαγραφή και εξαγωγή. Η προσθήκη ή επεξεργασία εγγράφου ενεργοποιεί μια δυναμική φόρμα, όπου καταγράφονται στοιχεία όπως όνομα υπαλλήλου, φύλο, ειδικότητα, ημερομηνίες και λοιπά πεδία μεταδεδομένων. Κατά την εξαγωγή, ο χρήστης έχει τη δυνατότητα προεπισκόπησης όλων των δεδομένων σε δομημένη μορφή, ενώ η αποστολή στο Google Apps Script γίνεται μέσω POST, στέλνοντας μόνο τα πεδία που απαιτεί το αντίστοιχο πρότυπο εγγράφου. Το παραγόμενο Google Doc αποθηκεύεται με κατάλληλο όνομα στο Google Drive και η σχετική πληροφορία καταγράφεται στο Firestore. Η όλη διαδικασία είναι οργανωμένη ώστε να προσφέρει real-time συγχρονισμό UI μέσω StreamBuilder, ασφάλεια δεδομένων ανά χρήστη (uid) και δυνατότητα άμεσης επέκτασης με νέα templates. Το Παράρτημα «Ροή Επεξεργασίας Εγγράφου» περιλαμβάνει αναλυτική παρουσίαση αυτής της διαδικασίας, συνοδευόμενη από εικόνες και στιγμιότυπα της εφαρμογής που τεκμηριώνουν τη διαδοχή βημάτων και τη ροή πληροφορίας από την επιλογή προτύπου έως την τελική αποθήκευση και εξαγωγή του εγγράφου.

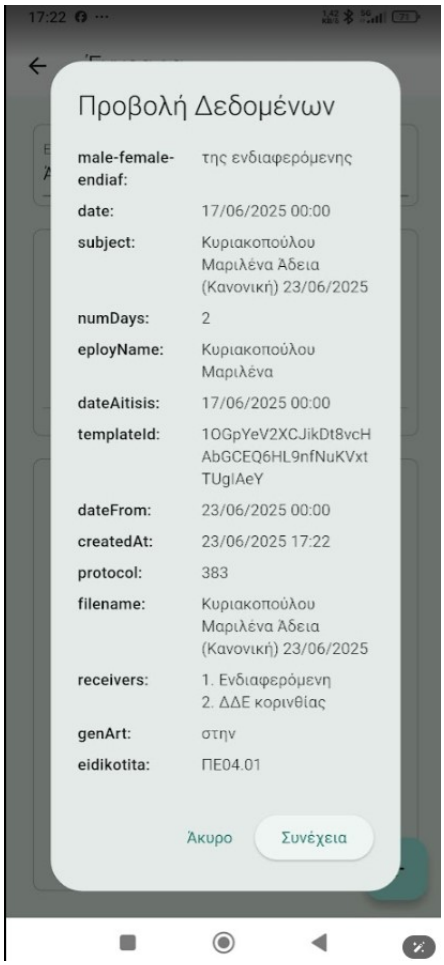
1) Επιλογή τύπου εγγράφου (template) – ο χρήστης ξεκινά επιλέγοντας από το drop-down τον τύπο εγγράφου που επιθυμεί να διαχειριστεί.

2)Προβολή υπάρχοντος εγγράφου – εμφανίζονται οι αποθηκευμένες εγγραφές σε κάρτες με επιλογές επεξεργασίας, διαγραφής και εξαγωγής.

3)Φόρμα συμπλήρωσης/επεξεργασίας – η αναδυόμενη φόρμα επιτρέπει την εισαγωγή νέων στοιχείων ή την επεξεργασία υπάρχοντος εγγράφου.



4)Προεπισκόπηση δεδομένων πριν την εξαγωγή – ο χρήστης έχει τη δυνατότητα να δει συγκεντρωτικά όλα τα πεδία σε διάλογο επιβεβαίωσης.



Η παραπάνω αλληλουχία παρουσιάζει τη συνολική ροή χρήστη, ξεκινώντας από την επιλογή template και φτάνοντας στην τελική εξαγωγή εγγράφου στο Google Drive.

Υλοποίηση

Ο κώδικας της υλοποίησης γίνεται στο αρχείο create_doc.dart, όπου ορίζεται η Stateful κλάση TemplateDropdownPage. Η οθόνη αυτή επιτρέπει στο χρήστη να επιλέγει τύπο εγγράφου από δυναμικά παραγόμενο dropdown, να προβάλλει υπάρχοντα έγγραφα από το Firestore και να τα επεξεργάζεται ή να προσθέτει νέα, μέσω διαλόγου φόρμας. Το StreamBuilder διατηρεί real-time συγχρονισμό με την Firestore συλλογή users/{uid}/templates/{templateId}/docs, αποδίδοντας αυτόματα τη λίστα εγγράφων όταν υπάρχουν αλλαγές.

Η φόρμα προσθήκης/επεξεργασίας υποστηρίζει πεδία όπως: όνομα υπαλλήλου, φύλο, ημερομηνίες, ειδικότητα, και subject/filename. Η διάδραση γίνεται αποκλειστικά με Material components (DropdownButtonFormField, TextFormField, AlertDialog, showDatePicker), ενώ η κατάσταση (state) διατηρείται τοπικά εντός της Stateful widget. Η υποβολή της φόρμας δημιουργεί ή ενημερώνει έγγραφα στο Firestore με χρήση add ή update, και κάθε λειτουργία συνοδεύεται από SnackBar ανατροφοδότηση.

Η εξαγωγή σε Google Doc υλοποιείται με POST κλήση σε Google Apps Script ή Firebase Cloud Function. Το template του εγγράφου ανακτάται από το φάκελο προτύπων στο Google Drive, και το όνομα του αρχείου προκύπτει αυτόματα βάσει πεδίου filename. Τα templates διαβάζονται δυναμικά από τον φάκελο της εφαρμογής, πράγμα που σημαίνει ότι μπορούν να προστεθούν νέοι τύποι εγγράφων χωρίς καμία αλλαγή στον κώδικα. Επιπλέον, κατά το export στέλνονται μόνο τα id και πεδία που χρησιμοποιεί το επιλεγμένο πρότυπο, εξασφαλίζοντας συμβατότητα και επεκτασιμότητα.

Στο Παράρτημα **(Δημιουργία Εγγράφων)** παρουσιάζεται λεπτομερής περιγραφή. Η παρακάτω εικόνα δείχνει ένα μέρος του κώδικα.

Διαχείριση ημερολογίου συμβάντων

Στοιχεία Αρχείου και Εισαγωγές

- **Όνομα** αρχείου**:** event_diary_page.dart
- **Σκοπός:** Διαχείριση ημερολογίου συμβάντων με δυνατότητες CRUD (Create, Read, Update, Delete) και εξαγωγής δεδομένων σε Google Sheets.
- **Βιβλιοθήκες:**
- **dart:convert:** Για JSON κωδικοποίηση/αποκωδικοποίηση
- **flutter/material.dart:** Για UI components
- **cloud_firestore:** Για σύνδεση με Firestore
- **http: Για HTTP** αιτήματα
- **mywidgets.dart, myfunctions.dart:** Προσαρμοσμένα components/συναρτήσεις
- **main.dart:** Χρήση της globalUid για προσδιορισμό χρήστη

Firestore Δομή Δεδομένων

```
CollectionReference<Map<String, dynamic>> _eventsRef() {
  if (globalUid == null) {
    throw Exception('Ο χρήστης δεν είναι συνδεδεμένος');
  }
  return FirebaseFirestore.instance
      .collection('users')
      .doc(globalUid)
      .collection('diaryEvents');
}
```

- **Ιεραρχία:**
  users → {userID} → diaryEvents → {eventDocs}
- **Πεδία εγγράφου:**
  event (string): Περιγραφή συμβάντος
- **date (Timestamp):** Ημερομηνία συμβάντος
- **createdAt/updatedAt** (Timestamp): Αυτόματη χρονοσφραγίδα.

Κύριο Widget: EventDiaryPage

```
return BorderedBox(
  child: ListView.separated(
    itemCount: docs.length,
    separatorBuilder: (_, __) => const Divider(height: 0),
    itemBuilder: (context, i) {
      final doc = docs[i];
      final data = doc.data();
      final dt = (data['date'] as Timestamp).toDate();

      return ListTile(
        //leading: const Icon(Icons.event),
        title: Text(
          '${dt.day.toString().padLeft(2, '0')}/${dt.month.toString().padLeft(2, '0')}/${dt.year}',
          style: const TextStyle(fontWeight: FontWeight.bold),
        ), // Text
        subtitle: Text(data['event'] ?? '–'),
        trailing: Row(
          mainAxisSize: MainAxisSize.min,
          children: [
            IconButton(
              icon: const Icon(Icons.edit, size: 20),
              onPressed: () => _showEditDialog(context, doc: doc),
            ), // IconButton
            IconButton(
              icon: const Icon(Icons.delete_forever_sharp, size: 20),
              onPressed: () => docs[i].reference.delete(),
            ), // IconButton
          ],
        ), // Row
      ); // ListTile
    },
  ), // ListView.separated
); // BorderedBox
```

Εξαγωγή σε Google Sheets

1. Έλεγχος ρυθμίσεων χρήστη:

```dart
Future<void> _exportToSheets(
  BuildContext context,
  DateTime firstDayOfYear,
) async {
  //final uid = devUid;
  final uid = globalUid;

  if (uid == null) {
    return;
  }
  const googleFolderKey = 'googleFolder';

  DocumentSnapshot<Map<String, dynamic>>? settingsSnap;
  String? folderId;

  try {
    settingsSnap =
        await FirebaseFirestore.instance
            .collection('users')
            .doc(uid)
            .collection('settings')
            .doc('app')
            .get();

    if (settingsSnap.exists) {
      final rawFolder = settingsSnap.data()?[googleFolderKey] as String?;

      if (rawFolder != null) {
        folderId = driveFolderIdFromUrl(rawFolder);
      }
    }
  } catch (e) {
```

2. Προϋποθέσεις:

```dart
if (folderId == null || folderId.trim().isEmpty) {
  if (context.mounted) {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(
        content: Text(
          ' ! Δεν έχει οριστεί φάκελος Drive. Ρύθμισέ τον πρώτα.',
        ), // Text
      ), // SnackBar
    );
  }
  return;
}

final snapshot =
    await _eventsRef()
        .orderBy('date')
        .where(
          'date',
          isGreaterThanOrEqualTo: Timestamp.fromDate(firstDayOfYear),
        )
        .get();

final records =
    snapshot.docs.map((doc) {
      final dt = (doc['date'] as Timestamp).toDate();
      return {
        'date':
            '${dt.day.toString().padLeft(2, "0")}/${dt.month.toString().padLeft(2, "0")}/$
        'event': doc['event'] ?? '',
      };
    }).toList();

if (records.isEmpty) {
  if (context.mounted) {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text(' ! Δεν υπάρχουν δεδομένα προς εξαγωγή')),
    );
  }
  return;
}
```

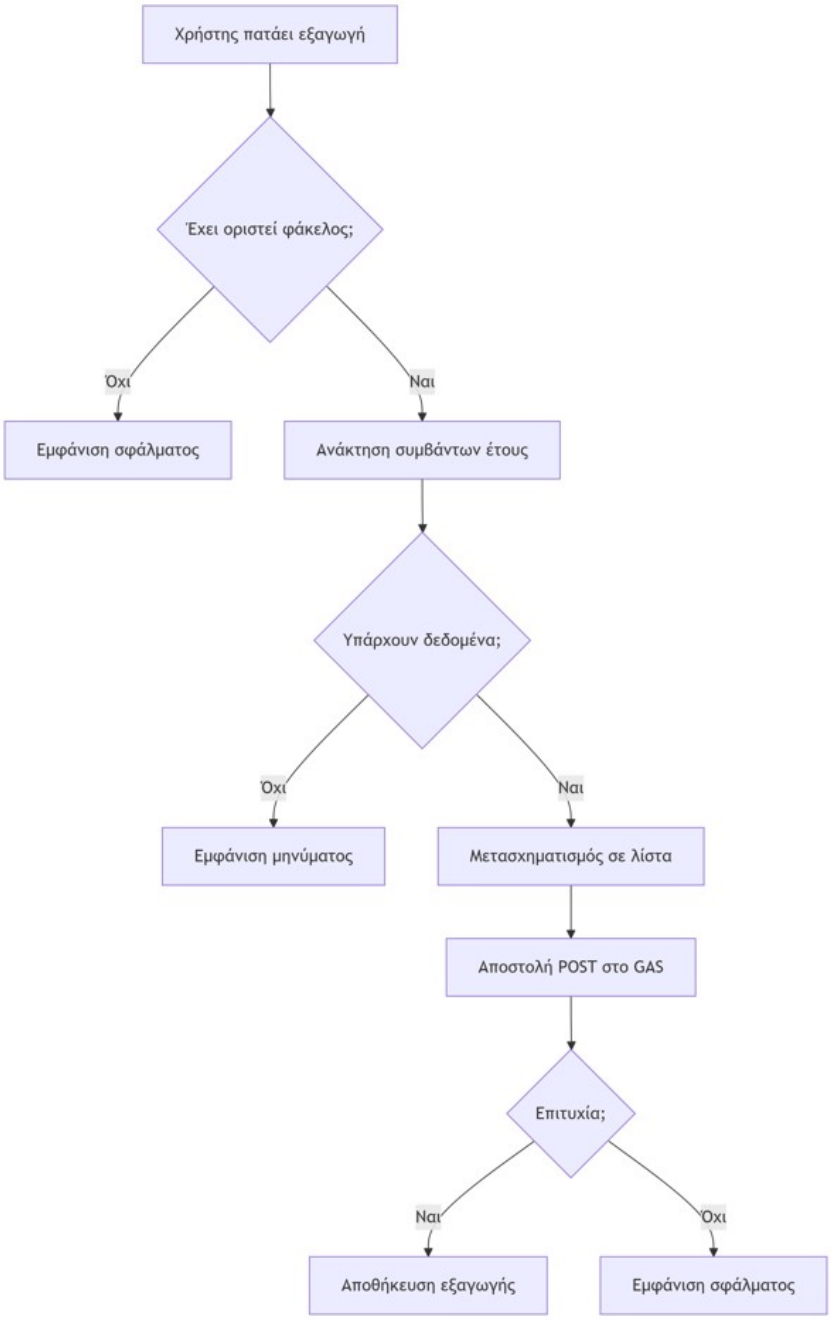
3. Αποστολή μέσω HTTP POST και Αντιμετώπιση Αποτελέσματος:

```dart
const webAppUrl = 'https://proxyexport-mhdemkezbq-uc.a.run.app';

try {
  final res = await http.post(
    Uri.parse(webAppUrl),
    headers: {'Content-Type': 'application/json'},
    body: jsonEncode({
      'uid': globalUid,
      'folderId': folderId,
      'records': records,
    }),
  );

  if (!context.mounted) return;
  final messenger = ScaffoldMessenger.of(context);

  if (res.statusCode == 200 && res.body.trim().isNotEmpty) {
    messenger.showSnackBar(
      const SnackBar(content: Text('✅ Εξαγωγή ολοκληρώθηκε!')),
    );

    await FirebaseFirestore.instance
        .collection('users')
        .doc(uid) // ----------------- uid
        .collection('exports')
        .add({'timestamp': FieldValue.serverTimestamp()});
  } else {
    throw Exception('Server error ${res.statusCode}: ${res.body}');
  }

} catch (e) {
  if (context.mounted) {
    ScaffoldMessenger.of(
      context,
    ).showSnackBar(SnackBar(content: Text('Σφάλμα εξαγωγής: $e')));
  }
}
}
}
```

Σημαντικά Σημεία Κώδικα

- Ασφάλεια: Έλεγχος globalUid != null πριν από Firestore operations
- Χρονοσφραγίδες: Αυτόματη καταγραφή createdAt/updatedAt
- Εξαγωγή μόνο τρέχοντος έτους: Φίλτρο isGreaterThanOrEqualTo: _firstDayOfYear
- UI/UX:
  - Real-time ενημέρωση μέσω StreamBuilder
  - Προσβασιμότητα (εικονίδια, μηνύματα)
  - Επεξεργασία χωρίς ανανέωση οθόνης

Διάγραμμα Ροής - Εξαγωγής



Πιθανές Βελτιώσεις

- Συμπληρωματικοί έλεγχοι σφαλμάτων (δικτύου, timeout)
- Δυνατότητα επιλογής εύρους ημερομηνιών
- Εισαγωγή από Excel/CSV
- Ταυτόχρονη εξαγωγή πολλών ετών
- Ενσωμάτωση ελέγχου ταυτότητας για το endpoint εξαγωγής

Πιλοτική Εφαρμογή και Αξιολόγηση

Περιγραφή σχολικού περιβάλλοντος δοκιμής

Η πιλοτική εφαρμογή του συστήματος πραγματοποιήθηκε στο Γυμνάσιο Ζευγολατιού και διήρκεσε δύο εβδομάδες. Η δοκιμή περιορίστηκε σε δύο χρήστες, τον Διευθυντή και την Υποδιευθύντρια του σχολείου, οι οποίοι αξιοποίησαν την εφαρμογή στο πλαίσιο της καθημερινής διοικητικής τους εργασίας.

Σενάρια χρήσης και ανατροφοδότηση

Κατά τη διάρκεια της πιλοτικής εφαρμογής, δοκιμάστηκαν όλες οι βασικές λειτουργίες του συστήματος, περιλαμβανομένων της δημιουργίας εγγράφων, της καταγραφής συμβάντων, της προβολής ιστορικού εγγράφων, καθώς και της εξαγωγής σε Google Docs και Sheets.

Ωστόσο, η λειτουργία που κρίθηκε ως η πιο ουσιαστική και χρήσιμη ήταν εκείνη της έκδοσης αδειών προσωπικού, η οποία μέχρι πρότινος απαιτούσε σημαντικό χρόνο και χειροκίνητη επεξεργασία εγγράφων. Η ανατροφοδότηση των χρηστών ήταν προφορική και δόθηκε σε άμεση συζήτηση κατά τη διάρκεια της δοκιμαστικής περιόδου.

Ανάλυση αποτελεσμάτων

Η συνολική εμπειρία των χρηστών υπήρξε θετική. Παρότι δεν συλλέχθηκαν ποσοτικά μετρήσιμα δεδομένα σχετικά με τον χρόνο συγγραφής ή επεξεργασίας εγγράφων, οι χρήστες ανέφεραν σημαντική βελτίωση στην οργάνωση των αρχείων, στην ταχύτητα εκτέλεσης διοικητικών εργασιών και, κυρίως, στη μείωση του άγχους διαχείρισης καθημερινών υποχρεώσεων. Η υποκειμενική αίσθηση ταχύτητας και η απλοποίηση των διαδικασιών αποτελούν ισχυρές ενδείξεις ότι το σύστημα επιτελεί τον στόχο του.

Περιορισμοί και ευκαιρίες βελτίωσης

Η πιλοτική δοκιμή δεν αποκάλυψε τεχνικά προβλήματα ή δυσλειτουργίες κατά τη χρήση της εφαρμογής. Ωστόσο, ο περιορισμένος αριθμός χρηστών και η μικρή διάρκεια δοκιμής αποτελούν περιορισμούς για την εξαγωγή γενικευμένων συμπερασμάτων.

Στο μέλλον, είναι σκόπιμο:

1. να επεκταθεί η εφαρμογή και σε γραμματειακό προσωπικό ή άλλους εκπαιδευτικούς,
2. να υλοποιηθεί ερωτηματολόγιο αξιολόγησης με ποσοτικά και ποιοτικά κριτήρια,
3. να ενσωματωθούν στατιστικά εργαλεία παρακολούθησης χρήσης (π.χ. χρόνος ανά ενέργεια, αριθμός εγγράφων),
4. να εξεταστεί η δυνατότητα offline λειτουργίας ή κινητής πρόσβασης μέσω mobile εφαρμογής.

Η πιλοτική εφαρμογή απέδειξε την πρακτική αξία και χρησιμότητα του συστήματος, δημιουργώντας ελπιδοφόρες προοπτικές για την ευρύτερη υιοθέτησή του στην εκπαιδευτική διοίκηση.

Συμπεράσματα και Μελλοντικές Επεκτάσεις

Συμπεράσματα

Η παρούσα εργασία ανέδειξε τη δυνατότητα αξιοποίησης σύγχρονων τεχνολογιών, όπως το Flutter, το Firebase και τα Google Apps Scripts, για την ανάπτυξη μιας εξατομικευμένης εφαρμογής γραμματειακής υποστήριξης διευθυντή σχολικής μονάδας.

Η εφαρμογή που σχεδιάστηκε και υλοποιήθηκε εξυπηρετεί βασικές ανάγκες διοικητικής διαχείρισης, όπως:

1. η δημιουργία και αρχειοθέτηση εγγράφων,
2. η παρακολούθηση ιστορικού ενεργειών,
3. η εξαγωγή δεδομένων σε Google Docs και Sheets,
4. και η διαχείριση αδειών εκπαιδευτικών.

Η πιλοτική εφαρμογή στο Γυμνάσιο Ζευγολατιού, παρότι περιορισμένη σε διάρκεια και συμμετέχοντες, προσέφερε ισχυρές ενδείξεις για τη χρηστικότητα, την ευκολία χρήσης και την πραγματική συμβολή του εργαλείου στην καθημερινή λειτουργία της σχολικής μονάδας.

Η θετική ανταπόκριση των χρηστών και η έλλειψη τεχνικών προβλημάτων ενισχύουν την άποψη ότι τέτοιου είδους λύσεις μπορούν να συμβάλουν ουσιαστικά στον εκσυγχρονισμό της εκπαιδευτικής διοίκησης.

Προτάσεις για μελλοντική εργασία

Η εμπειρία από την υλοποίηση και τη δοκιμαστική εφαρμογή του συστήματος ανέδειξε δυνατότητες και πεδία περαιτέρω εξέλιξης:

1. Εμπλουτισμός της βάσης προτύπων εγγράφων: Η δημιουργία επιπλέον κατηγοριών, όπως, αιτήσεις γονέων, διοικητικές αποφάσεις, πρακτικό συλλόγου θα ενισχύσει την αποδοτικότητα του συστήματος.
2. Υποστήριξη πολλαπλών ρόλων χρηστών: Ενσωμάτωση διαφορετικών επιπέδων πρόσβασης για γραμματείς, υποδιευθυντές και εκπαιδευτικούς γονείς.
3. Αναλυτικά στατιστικά χρήσης: Καταγραφή αριθμού εγγράφων, συχνότητας χρήσης, χρόνου ανά ενέργεια, για την αξιολόγηση της αποτελεσματικότητας.
4. Σύνδεση με το υπάρχον σύστημα MYSCHOOL για άντληση δεδομένων.

5. Ανάπτυξη web/desktop έκδοσης: Η παρούσα εφαρμογή είναι σχεδιασμένη για mobile συσκευές (Android), προσφέροντας ευελιξία χρήσης. Ωστόσο, για σταθερότερη ενσωμάτωση στη σχολική διοίκηση, προτείνεται η ανάπτυξη web-based ή Windows έκδοσης με δυνατότητα πρόσβασης από υπολογιστή γραφείου.

Τελικά σχόλια

Η εργασία αυτή αποτελεί ένα πρακτικό υπόδειγμα αξιοποίησης ψηφιακής τεχνολογίας για την υποστήριξη του έργου του Διευθυντή σχολείου. Αναδεικνύει πώς ακόμα και ένας μικρός αριθμός εργαλείων, όταν αξιοποιηθεί με σωστό σχεδιασμό και στόχευση, μπορεί να βελτιώσει σημαντικά την οργάνωση, την αποδοτικότητα και την ποιότητα του διοικητικού έργου στην εκπαίδευση.

Η προτεινόμενη εφαρμογή δεν φιλοδοξεί να αντικαταστήσει ολοκληρωμένα συστήματα, αλλά να λειτουργήσει ως συμπληρωματικό και ευέλικτο εργαλείο, προσαρμοσμένο στις ανάγκες του ίδιου του Διευθυντή. Τα ευρήματα της παρούσας μελέτης, αν αξιοποιηθούν σωστά, μπορούν να αποτελέσουν τη βάση για περαιτέρω τεχνολογική ενσωμάτωση και καινοτομία στο πεδίο της σχολικής διοίκησης.

Παραρτήματα (Appendix)

Α) Αρχεία της εφαρμογής

Ο κώδικας της εφαρμογής αναπτύχθηκε στα παρακάτω αρχεία

Φάκελος / Αρχείο	Ρόλος & περιεχόμενο
generated/intl/l10n.dart	Αυτόματο αρχείο που δημιουργεί το `intl_utils`· περιέχει τα classes/lookup για τις μεταφράσεις.
l10n/app_en.arb	Αγγλικές συμβολοσειρές
l10n/firebase_ui_el.arb	Ελληνικές μεταφράσεις για τα έτοιμα widgets του Firebase UI Auth.
l10n/intl_en.arb	Πρότυπο (.arb-template) που παράγει ο generator· λειτουργεί ως fallback/πηγή νέων κλειδιών.
mylib/drive_service.dart	Helper-service για κλήσεις στο Google Drive API (list, upload, download).
mylib/firebase_options.dart	Αυτο-παράγεται από το `flutterfire configure`· περιέχει τα Firebase API-keys & IDs για κάθε πλατφόρμα.
mylib/greek_localizations.dart	Πρόσθετα strings στα Ελληνικά (δικά μου μηνύματα ή override υπάρχοντων).
mylib/my_greek_delegate.dart	`LocalizationsDelegate` που «δένει» τα δικά μου Greek strings με το Flutter framework.
mylib/myfunctions.dart	Γενικές βοηθητικές συναρτήσεις (π.χ. date formatters, validators, converters).
mylib/mywidgets.dart	Επαναχρησιμοποιούμενα δικά μου custom widgets για ομοιομορφία UI.
mylib/providers.dart	Ορισμοί Riverpod providers (state-management: auth state, ρυθμίσεις, Firestore streams κ.λπ.).
src/appointments_page.dart	Οθόνη διαχείρισης/προβολής ραντεβού.
src/create_doc.dart	Σελίδα δημιουργίας & upload εγγράφου
src/event_diary_page.dart	Ημερολόγιο συμβάντων — λίστα & λεπτομέρειες.
src/main_menu.dart	Κεντρικό menu/navigation hub της εφαρμογής.
src/main.dart	Entry point: `void main()`, αρχικοποίηση Firebase, `MaterialApp`, routing.
src/screen_settings.dart	Οθόνη ρυθμίσεων χρήστη
pubspec.yalm	εξαρτήσεις και οι βιβλιοθήκες που απαιτήθηκαν στην ανάπτυξη της εφαρμογής

B) Firestore

Εισαγωγή

Το Firestore είναι μια προηγμένη cloud-based NoSQL βάση δεδομένων, τμήμα της πλατφόρμας Firebase της Google. Είναι σχεδιασμένο ειδικά για εφαρμογές κινητών συσκευών και web, παρέχοντας δυνατότητες συγχρονισμού και αποθήκευσης δεδομένων σε πραγματικό χρόνο. Το Firestore υποστηρίζει offline αποθήκευση δεδομένων και υψηλή κλιμάκωση, προσφέροντας ευελιξία σε ανάπτυξη και συντήρηση εφαρμογών.

Βασικές Έννοιες και Δομή του Firestore

Το Firestore αποθηκεύει τα δεδομένα του με μια ιεραρχική δομή βασισμένη σε **συλλογές (collections)** και **έγγραφα (documents)**. Οι **συλλογές** αποτελούν ομάδες από έγγραφα, και κάθε έγγραφο περιέχει δεδομένα οργανωμένα σε μορφή **ζευγών κλειδιού-τιμής**, δηλαδή σε δομή τύπου JSON.

Παράδειγμα:

users (Collection)

userID1 (Document)

name: "Δημήτρης"

email: "dimitris@example.com"

userID2 (Document)

name: "Αλκιβιάδης"

email: "alkis@example.com"

Κάθε έγγραφο μπορεί να περιέχει πληροφορίες όπως όνομα, email, ηλικία, ημερομηνία εγγραφής, καθώς και πιο σύνθετα δεδομένα όπως πίνακες ή αντικείμενα. Τα δεδομένα αυτά αποθηκεύονται σε **πεδία (fields)**, τα οποία μπορούν να περιλαμβάνουν:

- Συμβολοσειρές (π.χ. ονόματα ή διευθύνσεις email)
- Αριθμούς (π.χ. ηλικία ή αριθμούς τηλεφώνου)
- Τιμές boolean (true/false)
- Ημερομηνίες (με timestamp)
- Πίνακες (arrays)
- Αντικείμενα (nested objects)
- Αναφορές σε άλλα έγγραφα (document references)
- Γεωγραφικές συντεταγμένες (GeoPoint)

Η ευελιξία αυτή επιτρέπει στον προγραμματιστή να δομήσει τη βάση δεδομένων με τρόπο φυσικό για την εφαρμογή του, διατηρώντας παράλληλα την ικανότητα κλιμάκωσης και τον συγχρονισμό σε πραγματικό χρόνο που προσφέρει το Firestore.

Εγκατάσταση και Σύνδεση με την Εφαρμογή

Προαπαιτούμενα Εργαλεία και Τεχνολογίες

Πριν από την ενσωμάτωση του Firebase σε μια εφαρμογή Flutter, είναι αναγκαία η ύπαρξη συγκεκριμένων τεχνολογικών προϋποθέσεων στο περιβάλλον ανάπτυξης. Συγκεκριμένα, απαιτούνται:

- Flutter SDK έκδοση 3.3 ή μεταγενέστερη (επιβεβαιώνεται με την εντολή flutter --version).
- Dart SDK έκδοση 3.2 ή νεότερη.
- Λογαριασμός Firebase, προσβάσιμος μέσω της πλατφόρμας https://console.firebase.google.com.
- Node.js και npm, απαραίτητα για την εγκατάσταση και λειτουργία του εργαλείου Firebase CLI.
- Git, προαιρετικά, για την κλωνοποίηση αποθετηρίων και παραδειγμάτων εφαρμογών από πλατφόρμες όπως το GitHub.

Η ικανοποίηση των παραπάνω προϋποθέσεων διασφαλίζει την ομαλή λειτουργία του συστήματος ανάπτυξης και την επιτυχή σύνδεση της εφαρμογής με τις υπηρεσίες του Firebase.

Δημιουργία Έργου στο Firebase Console

Η αρχική διαμόρφωση πραγματοποιείται μέσω του **Firebase Console**. Ο προγραμματιστής συνδέεται στον λογαριασμό του και επιλέγει **"Add project"** για τη δημιουργία ενός νέου έργου. Κατά τη διαδικασία:

1. Καθορίζεται ένα όνομα έργου
2. Η χρήση του Google Analytics μπορεί να απενεργοποιηθεί, καθώς δεν είναι απαραίτητη για όλες τις εφαρμογές.
3. Με την ολοκλήρωση της διαδικασίας, δημιουργείται το βασικό περιβάλλον για την εφαρμογή.

Στη συνέχεια, προστίθενται οι πλατφόρμες Android και iOS:

- Για το Android δηλώνεται το package name, π.χ. com.example.chatdemo.
- Για το iOS καθορίζεται το Bundle ID, π.χ. com.example.chatdemo.

Σημειώνεται ότι δεν είναι απαραίτητη η χειροκίνητη λήψη των αρχείων google-services.json και GoogleService-Info.plist, καθώς αυτά δημιουργούνται αυτόματα μέσω της χρήσης του FlutterFire CLI.

Εγκατάσταση Εργαλείων CLI (Command Line Interface)

Για την αλληλεπίδραση του τοπικού περιβάλλοντος με το Firebase απαιτείται η εγκατάσταση δύο εργαλείων: του Firebase CLI και του FlutterFire CLI. Η διαδικασία ξεκινά από το τερματικό (PowerShell, Command Prompt ή Git Bash), εκτός του φακέλου του έργου, με τις παρακάτω εντολές:

npm install -g firebase-tools
firebase login
dart pub global activate flutterfire_cli

Η πρώτη εντολή εγκαθιστά το Firebase CLI, ενώ η δεύτερη πραγματοποιεί σύνδεση του λογαριασμού Firebase στο τοπικό περιβάλλον. Η τρίτη εντολή ενεργοποιεί το FlutterFire CLI, το οποίο θα χρησιμοποιηθεί για τη σύνδεση του Flutter project με το Firebase project.

Για τη σωστή λειτουργία του FlutterFire CLI, είναι απαραίτητο να περιλαμβάνεται η διαδρομή $HOME/.pub-cache/bin στη μεταβλητή περιβάλλοντος PATH.

Αφού εισέλθουμε στον φάκελο του Flutter project, εκτελούμε:

flutterfire configure

Το εργαλείο αναγνωρίζει το έργο, ζητά επιλογή των υποστηριζόμενων πλατφορμών (Android, iOS, Web) και παράγει αυτόματα το αρχείο firebase_options.dart με όλες τις απαραίτητες ρυθμίσεις σύνδεσης.

Προσθήκη στο pubspec.yaml

Για τη χρήση των βασικών υπηρεσιών του Firebase, είναι απαραίτητη η προσθήκη των σχετικών πακέτων εξαρτήσεων στο αρχείο pubspec.yaml. Οι σταθερές εκδόσεις κατά το χρόνο υλοποίησης (Μάιος 2025) είναι οι εξής:

```
dependencies:
 firebase_core: ^3.6.0
 cloud_firestore: ^5.6.4
```

Η εντολή flutter pub get αναλαμβάνει να κατεβάσει και να ενσωματώσει τα πακέτα στο έργο.

Αρχικοποίηση στο main.dart

Η αρχικοποίηση του Firebase πραγματοποιείται στο αρχείο main.dart, πριν την εκκίνηση της εφαρμογής. Ο σχετικός κώδικας έχει ως εξής:

```
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'firebase_options.dart';

Future<void> main() async {
	WidgetsFlutterBinding.ensureInitialized();
	await Firebase.initializeApp(
		options: DefaultFirebaseOptions.currentPlatform,
	);
	runApp(const MyApp());
}
```

Ο κώδικας διασφαλίζει ότι, πριν την εκκίνηση της εφαρμογής (runApp), έχει ολοκληρωθεί η αρχικοποίηση του Firebase με βάση τις παραμέτρους που παρέχονται στο αρχείο firebase_options.dart. Το αρχείο αυτό δημιουργείται αυτόματα από το εργαλείο flutterfire configure και περιλαμβάνει τα στοιχεία σύνδεσης για τις δηλωμένες πλατφόρμες.

Εφόσον ολοκληρωθούν όλα τα παραπάνω βήματα, η εφαρμογή είναι πλήρως συνδεδεμένη με το Firebase και μπορεί να αξιοποιήσει τις παρεχόμενες υπηρεσίες του, όπως αποθήκευση δεδομένων, αυθεντικοποίηση, cloud messaging και άλλες.

Authentication Flutter + Firebase (Firestore)

Δημιουργία Project στο Firebase Console

Η πρώτη φάση της ενσωμάτωσης του Firebase σε μια εφαρμογή αφορά τη δημιουργία ενός νέου project μέσω της πλατφόρμας Firebase Console. Για τον σκοπό αυτό, ο χρήστης μεταβαίνει στη διαδικτυακή διεύθυνση https://console.firebase.google.com, όπου έχει τη δυνατότητα να δημιουργήσει ένα νέο έργο επιλέγοντας "Add project" ή "Δημιουργία νέου έργου". Κατά την έναρξη της διαδικασίας, καλείται να ορίσει ένα όνομα για το project, το οποίο μπορεί να είναι ενδεικτικό της εφαρμογής (π.χ. MyFlutterApp), και να αποδεχθεί τους όρους χρήσης της υπηρεσίας. Η δημιουργία του έργου αποτελεί βασική προϋπόθεση για την ενεργοποίηση των επιμέρους υπηρεσιών του Firebase, όπως η ταυτοποίηση χρηστών (Authentication), η αποθήκευση δεδομένων (Cloud Firestore), και άλλες λειτουργίες που θα αξιοποιηθούν στο πλαίσιο της παρούσας εφαρμογής.

Προσθήκη Πλατφορμών (Android, Web)

Κατά την υλοποίηση της διασύνδεσης της εφαρμογής με την πλατφόρμα Firebase, καθοριστικής σημασίας στάδιο αποτελεί η προσθήκη των επιμέρους πλατφορμών, όπως Android και Web, μέσω του Firebase Console. Για την προσθήκη της πλατφόρμας Android, ο χρήστης μεταβαίνει στο περιβάλλον διαχείρισης του Firebase project και επιλέγει "Add app", κάνοντας κλικ στο αντίστοιχο εικονίδιο της πλατφόρμας Android. Στη συνέχεια, ζητείται η συμπλήρωση του Android package name, ο οποίος εντοπίζεται στο αρχείο AndroidManifest.xml της εφαρμογής, στη διαδρομή android/app/src/main/. Προαιρετικά μπορεί να δηλωθεί και ένα App nickname, για σκοπούς εσωτερικής αναγνώρισης.

Απαραίτητο στοιχείο της καταχώρισης αποτελεί η υποβολή των πιστοποιητικών SHA-1 και SHA-256, τα οποία απαιτούνται για την υποστήριξη λειτουργιών όπως η ταυτοποίηση χρηστών μέσω Google Sign-In. Για την εξαγωγή των τιμών αυτών, ο χρήστης μπορεί να αξιοποιήσει το ενσωματωμένο τερματικό του Visual Studio Code, μεταβαίνοντας αρχικά στον φάκελο android του έργου με την εντολή cd android, και στη συνέχεια εκτελώντας την εντολή ./gradlew signingReport. Με την ολοκλήρωση της διαδικασίας, στην έξοδο του τερματικού εμφανίζονται τα αποτυπώματα SHA-1 και SHA-256 κάτω από τα αντίστοιχα build variants (debug ή release), τα οποία και επικολλούνται στο σχετικό πεδίο του Firebase Console.

Ολοκληρώνοντας τη διαδικασία, παρέχεται προς λήψη το αρχείο διαμόρφωσης google-services.json, το οποίο πρέπει να τοποθετηθεί στον φάκελο android/app/ του Flutter project. Το αρχείο αυτό αποτελεί βασικό στοιχείο για τη σύνδεση της εφαρμογής με το Firebase backend, καθώς περιέχει τα απαραίτητα αναγνωριστικά και ρυθμίσεις για τη σωστή λειτουργία της υπηρεσίας.

Ενσωμάτωση Firebase SDK στο Flutter project

Στο πλαίσιο της ενσωμάτωσης του Firebase στο περιβάλλον ανάπτυξης Flutter, κρίσιμο στάδιο αποτελεί η τροποποίηση των αρχείων διαμόρφωσης του έργου, προκειμένου να καταστεί δυνατή η αξιοποίηση των υπηρεσιών του Firebase SDK. Συγκεκριμένα, στο αρχείο build.gradle που βρίσκεται στον ριζικό φάκελο android/, απαιτείται η προσθήκη της εντολής classpath 'com.google.gms:google-services:4.3.15' στο τμήμα buildscript > dependencies, προκειμένου να ενεργοποιηθεί το plugin της Google για τις υπηρεσίες Firebase. Παράλληλα, στο αρχείο build.gradle του φακέλου android/app/, είναι αναγκαία η προσθήκη της γραμμής apply plugin: 'com.google.gms.google-services' στο τέλος του αρχείου, ώστε να ενεργοποιηθεί η υποστήριξη για τη λειτουργικότητα των Google services. Επιπλέον, στο τμήμα dependencies του ίδιου αρχείου, εισάγεται η γραμμή implementation platform('com.google.firebase:firebase-bom:32.3.1'), η οποία εξασφαλίζει τη διαχείριση συμβατών εκδόσεων μεταξύ των επιμέρους βιβλιοθηκών του Firebase. Οι παρεμβάσεις αυτές αποτελούν αναγκαία προαπαιτούμενα για την απρόσκοπτη λειτουργία των υπηρεσιών Firebase, όπως η ταυτοποίηση χρηστών, η αποθήκευση δεδομένων στο Firestore και η χρήση αναλυτικών στοιχείων.Εργαλεία γραμμής εντολών (FlutterFire CLI)

Εγκατάσταση πακέτων Flutter

Στο επόμενο στάδιο, καθίσταται απαραίτητη η εγκατάσταση των κατάλληλων πακέτων (packages) που επιτρέπουν την ενσωμάτωση του Firebase στο περιβάλλον του Flutter. Η διαδικασία πραγματοποιείται μέσω του διαχειριστή πακέτων του Flutter και εκτελείται από το ενσωματωμένο τερματικό του Visual Studio Code. Συγκεκριμένα, εγκαθίστανται τα ακόλουθα πακέτα:

- **firebase_core**: το βασικό πακέτο που απαιτείται για την αρχικοποίηση του Firebase στην εφαρμογή.
- **firebase_auth**: πακέτο υπεύθυνο για τη διαχείριση της αυθεντικοποίησης χρηστών.
- **cloud_firestore**: επιτρέπει την πρόσβαση και αλληλεπίδραση με τη βάση δεδομένων Firestore.
- **firebase_ui_auth**: προσφέρει έτοιμα widgets για login/registration μέσω Firebase.
- **firebase_ui_localizations**: υποστηρίζει την τοπικοποίηση των UI στοιχείων αυθεντικοποίησης.
- **firebase_ui_oauth_google**: ενεργοποιεί τη σύνδεση μέσω Google λογαριασμού.

Η εγκατάσταση των πακέτων αυτών πραγματοποιείται με τις αντίστοιχες εντολές στο τερματικό:

```
flutter pub add firebase_core
flutter pub add firebase_auth
flutter pub add cloud_firestore
flutter pub add firebase_ui_auth
```

flutter pub add firebase_ui_localizations
flutter pub add firebase_ui_oauth_google

Η επιτυχής ολοκλήρωση του βήματος αυτού εξασφαλίζει ότι η εφαρμογή διαθέτει όλα τα απαραίτητα εξαρτήματα για την αλληλεπίδραση με τις βασικές υπηρεσίες του Firebase, επιτρέποντας την περαιτέρω ανάπτυξη λειτουργιών αυθεντικοποίησης και αποθήκευσης δεδομένων.

FlutterFire CLI

Στο πλαίσιο της διασύνδεσης της εφαρμογής με το Firebase, το Βήμα 5 περιλαμβάνει τη χρήση του εργαλείου FlutterFire CLI (Command Line Interface), το οποίο αυτοματοποιεί τη διαδικασία σύνδεσης ενός Flutter project με ένα Firebase project. Η χρήση του εργαλείου αυτού επιτρέπει τη δημιουργία αρχείων διαμόρφωσης που είναι απαραίτητα για την αρχικοποίηση και τη σωστή λειτουργία των υπηρεσιών Firebase μέσα στην εφαρμογή.

Αρχικά, απαιτείται η εγκατάσταση του FlutterFire CLI με την εντολή:

dart pub global activate flutterfire_cli

Μετά την εγκατάσταση, είναι σημαντικό να διασφαλιστεί ότι το εκτελέσιμο αρχείο περιλαμβάνεται στη διαδρομή συστήματος (PATH). Αυτό επιτυγχάνεται με την προσθήκη της διαδρομής $HOME/.pub-cache/bin στο αρχείο ρυθμίσεων του shell (.bashrc, .zshrc ή άλλο ανάλογα με το λειτουργικό σύστημα).

Στη συνέχεια, η εντολή flutterfire configure εκτελείται από τον ριζικό φάκελο του έργου. Το εργαλείο καθοδηγεί τον χρήστη ώστε να επιλέξει το αντίστοιχο Firebase project και να δηλώσει τις πλατφόρμες που πρόκειται να υποστηριχθούν (π.χ. Android, iOS, Web). Με την ολοκλήρωση της διαδικασίας, δημιουργείται αυτόματα το αρχείο firebase_options.dart στον φάκελο lib/, το οποίο περιλαμβάνει τις απαραίτητες παραμέτρους σύνδεσης με το backend του Firebase.

Η αξιοποίηση του FlutterFire CLI επιταχύνει τη διαδικασία ενσωμάτωσης και μειώνει τον κίνδυνο σφαλμάτων από χειροκίνητες ρυθμίσεις, καθιστώντας το ένα ιδιαίτερα χρήσιμο εργαλείο στο πλαίσιο ανάπτυξης εφαρμογών που βασίζονται στο Firebase.

Authentication στο Firebase Console

Στο πλαίσιο της υλοποίησης μηχανισμών αυθεντικοποίησης χρηστών στην εφαρμογή, η ρύθμιση της υπηρεσίας Authentication στο Firebase Console αποτελεί ένα κρίσιμο βήμα. Μέσω του Authentication, η εφαρμογή αποκτά τη δυνατότητα να διαχειρίζεται με ασφαλή τρόπο διαδικασίες εγγραφής, σύνδεσης και ταυτοποίησης χρηστών με διάφορους τρόπους (email, Google, Apple κ.ά.).

Για να ενεργοποιηθεί η υπηρεσία, ο χρήστης μεταβαίνει στο Firebase Console και επιλέγει από το πλαϊνό μενού την ενότητα "Authentication". Στη συνέχεια, επιλέγει την καρτέλα "Sign-in method", όπου εμφανίζεται λίστα με όλους τους διαθέσιμους παρόχους σύνδεσης. Σε πρώτη φάση, είναι απαραίτητη η ενεργοποίηση της μεθόδου Email/Password, η οποία επιτρέπει στους χρήστες να δημιουργούν λογαριασμό με τη χρήση διεύθυνσης ηλεκτρονικού ταχυδρομείου και κωδικού πρόσβασης. Η μέθοδος

αυτή αποτελεί τη βασική και πιο ευρέως χρησιμοποιούμενη επιλογή για την είσοδο σε εφαρμογές Flutter που βασίζονται στο Firebase.

Προαιρετικά, μπορεί να ενεργοποιηθεί και η σύνδεση μέσω Google Sign-In ή Apple Sign-In, εφόσον η εφαρμογή απαιτεί κοινωνική ή ταχύτερη είσοδο χρηστών. Στην περίπτωση ενεργοποίησης Google Sign-In, απαιτείται η δημιουργία OAuth 2.0 Client ID μέσω του Google Cloud Console και η αντιστοίχιση του στο Firebase project. Η ενσωμάτωση αυτών των μεθόδων προσφέρει βελτιωμένη εμπειρία χρήστη και αυξημένη ασφάλεια.

Η ρύθμιση του Authentication στο Firebase αποτελεί δομικό στοιχείο για την υποστήριξη ασφαλούς πρόσβασης στις λειτουργίες της εφαρμογής και καθιστά δυνατή την εξατομίκευση των υπηρεσιών που προσφέρει το λογισμικό, βάσει του ταυτοποιημένου προφίλ κάθε χρήστη.

Προσθήκη Εξαρτήσεων (Dependencies) στο pubspec.yaml

Για την ολοκληρωμένη λειτουργία της εφαρμογής σε περιβάλλον Firebase, απαιτείται η προσθήκη των απαραίτητων εξαρτήσεων στο αρχείο pubspec.yaml, το οποίο αποτελεί το βασικό αρχείο διαχείρισης πακέτων στο οικοσύστημα του Flutter. Οι εξαρτήσεις αυτές παρέχουν τη δυνατότητα πρόσβασης σε υπηρεσίες του Firebase, όπως η αυθεντικοποίηση χρηστών και η αποθήκευση δεδομένων σε cloud βάση.

Ενδεικτικά, η διαμόρφωση του αρχείου pubspec.yaml, με βάση τις σταθερές εκδόσεις που υποστηρίζονται από το Firebase BoM (Bill of Materials) έκδοση 3.11.0 (Μάιος 2025), είναι η ακόλουθη:

```
dependencies:
 flutter:
   sdk: flutter
 firebase_core: ^3.11.0        # απαραίτητο για αρχικοποίηση Firebase
 firebase_auth: ^5.11.0        # διαχείριση αυθεντικοποίησης χρηστών
 cloud_firestore: ^6.4.0       # πρόσβαση στη βάση δεδομένων Firestore
 firebase_ui_auth: ^1.14.0     # (προαιρετικό) έτοιμα UI components για login/register
```

Η έκδοση 3.11.0 του **Flutter BoM** περιλαμβάνει τις πλέον πρόσφατες εκδόσεις πακέτων, συμβατές μεταξύ τους, ενώ υποστηρίζει και νέες τεχνολογίες ταυτοποίησης όπως τα **Passkeys**, προσφέροντας αυξημένη ασφάλεια και συμβατότητα με σύγχρονες πλατφόρμες.
Η ενσωμάτωση των παραπάνω εξαρτήσεων γίνεται με την εντολή:

flutter pub get

η οποία φροντίζει για την εγκατάσταση και διασύνδεση των δηλωμένων πακέτων στο περιβάλλον ανάπτυξης της εφαρμογής.

Αρχικοποίηση στο main.dart

Μετά την εγκατάσταση των πακέτων και τη διαμόρφωση των αρχείων σύνδεσης, απαιτείται η αρχικοποίηση του Firebase κατά την εκκίνηση της εφαρμογής. Η ενέργεια αυτή διασφαλίζει ότι όλες οι υπηρεσίες του Firebase είναι διαθέσιμες και λειτουργικές από το πρώτο frame της εφαρμογής.

Η αρχικοποίηση πραγματοποιείται στο αρχείο main.dart, το οποίο αποτελεί το σημείο εισόδου κάθε εφαρμογής Flutter. Ο σχετικός κώδικας έχει ως εξής:

```dart
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();

  await Firebase.initializeApp(options: DefaultFirebaseOptions.currentPlatform);

  FirebaseFirestore.instance.settings = const Settings(
    persistenceEnabled: true,
    cacheSizeBytes: Settings.CACHE_SIZE_UNLIMITED,
  );

  devUid = 'oRP8yo3mJcPF03N5kAWfgEwZaj03';

  // ▼▼▼ ΕΔΩ: ProviderScope ψηλότερα απ' όλα ▼▼▼
  runApp(const ProviderScope(child: MyApp()));
}
```

Η εντολή WidgetsFlutterBinding.ensureInitialized() εξασφαλίζει ότι το Flutter περιβάλλον έχει φορτωθεί πλήρως πριν την εκκίνηση ασύγχρονων λειτουργιών, όπως η αρχικοποίηση του Firebase. Στη συνέχεια, η Firebase.initializeApp() καλείται με το αντικείμενο DefaultFirebaseOptions.currentPlatform, το οποίο περιέχει τις παραμέτρους σύνδεσης που έχουν δημιουργηθεί από το εργαλείο flutterfire configure και αντιστοιχούν στην πλατφόρμα στην οποία εκτελείται η εφαρμογή (Android, iOS ή Web).

Η επιτυχημένη εκτέλεση της παραπάνω συνάρτησης καθιστά διαθέσιμες όλες τις υπηρεσίες Firebase καθ' όλη τη διάρκεια ζωής της εφαρμογής και αποτελεί απαραίτητη προϋπόθεση για την απρόσκοπτη χρήση λειτουργιών όπως η αυθεντικοποίηση, η αποθήκευση δεδομένων ή η ειδοποίηση χρηστών μέσω cloud messaging.

Υλοποίηση Έτοιμης Ροής Αυθεντικοποίησης με FirebaseUI

Η βιβλιοθήκη **FirebaseUI for Flutter** παρέχει ένα σύνολο έτοιμων και προσαρμόσιμων διεπαφών χρήστη για τη διαχείριση της αυθεντικοποίησης, ελαχιστοποιώντας σημαντικά την ανάγκη για χειροκίνητη κατασκευή login οθονών. Η ενσωμάτωση του FirebaseUI επιτρέπει την ταχεία δημιουργία οθονών για **είσοδο, εγγραφή, επαναφορά κωδικού πρόσβασης**, καθώς και υποστήριξη **πολλαπλών παρόχων ταυτοποίησης** όπως email, Google, Apple κ.ά.

Η παρακάτω υλοποίηση, μέσω μόλις δύο γραμμών κώδικα σε MaterialApp, προσφέρει πλήρη ροή αυθεντικοποίησης, η οποία περιλαμβάνει:

- Είσοδο με email
- Αυτόματη πλοήγηση στην κύρια σελίδα της εφαρμογής μετά την επιτυχή σύνδεση

Η χρήση του SignInScreen μέσω FirebaseUI παρέχει προκαθορισμένες επιλογές για **εγγραφή νέου χρήστη (Register)**, **επαναφορά κωδικού (Forgot Password)** και διαχείριση σφαλμάτων, χωρίς να απαιτείται πρόσθετη λογική από τον προγραμματιστή.
Η Google παρέχει επίσης εκτενές **codelab** με αναλυτικό παράδειγμα πλήρους εφαρμογής βασισμένης στο FirebaseUI, η οποία περιλαμβάνει την προσαρμογή της εμφάνισης, την υποστήριξη localization, καθώς και εναλλακτικούς τρόπους ροής πλοήγησης και διαχείρισης κατάστασης σύνδεσης χρηστών.

```
79      return MaterialApp(
80        title: 'School Admin',
81        debugShowCheckedModeBanner: false,
82
83        supportedLocales: const [Locale('el'), Locale('en')],
84        localizationsDelegates: const [
85          FirebaseUILocalizations.delegate,
86          GlobalMaterialLocalizations.delegate,
87          GlobalWidgetsLocalizations.delegate,
88          GlobalCupertinoLocalizations.delegate,
89          //GreekLocalizationsDelegate(),
90        ],
91
92        theme: ThemeData(
93          colorScheme: ColorScheme.fromSeed(seedColor: Colors.teal),
94          useMaterial3: true,
95        ).copyWith(scaffoldBackgroundColor: const Color(0xFFE8F5E9)),
96
97        home: StreamBuilder<auth.User?>(
98          stream: auth.FirebaseAuth.instance.authStateChanges(),
99          builder: (context, snapshot) {
100           if (snapshot.connectionState == ConnectionState.waiting) {
101             return const Scaffold(
102               body: Center(child: CircularProgressIndicator()),
103             ); // Scaffold
104           }
105
106           if (snapshot.hasData) {
107             _saveUserToFirestore(snapshot.data!);
108             return const HomeScreen();
109           }
```

Η υιοθέτηση του FirebaseUI στη συγκεκριμένη εφαρμογή επιτρέπει την εστίαση της ανάπτυξης στις λειτουργικές απαιτήσεις του συστήματος, παρέχοντας παράλληλα μια ασφαλή, αξιόπιστη και συμβατή με τις προδιαγραφές του Firebase λύση αυθεντικοποίησης.

Αποθήκευση Προφίλ Χρήστη στο Cloud Firestore

Μετά την επιτυχή διαδικασία αυθεντικοποίησης ενός χρήστη, είναι συχνά απαραίτητο να αποθηκευτούν επιπλέον πληροφορίες προφίλ που δεν παρέχονται απευθείας από το Firebase Authentication, όπως το προβαλλόμενο όνομα, η ημερομηνία εγγραφής, προτιμήσεις ή άλλα μεταδεδομένα που σχετίζονται με τη χρήση της εφαρμογής. Η

ενδεδειγμένη προσέγγιση για την αποθήκευση αυτών των δεδομένων είναι η χρήση του Cloud Firestore, της κλιμακούμενης βάσης δεδομένων NoSQL του Firebase.

Ο παρακάτω κώδικας δείχνει τη διαδικασία αποθήκευσης ενός βασικού προφίλ χρήστη:

```
124                 AuthStateChangeAction<SignedIn>((context, state) async {
125                   final user = auth.FirebaseAuth.instance.currentUser;
126                   if (user != null) await _saveUserToFirestore(user);
127                   if (context.mounted) {
128                     Navigator.pushReplacement(
129                       context,
130                       MaterialPageRoute(
131                         builder: (_) => const HomeScreen(),
132                       ), // MaterialPageRoute
133                     );
134                   }
135                 }), // AuthStateChangeAction
```

Η μεταβλητή uid ανακτάται από τον τρέχοντα ταυτοποιημένο χρήστη, ο οποίος παρέχεται από την υπηρεσία Firebase Authentication. Το UID χρησιμοποιείται ως μοναδικό αναγνωριστικό εγγράφου εντός της συλλογής users στο Firestore, διασφαλίζοντας έτσι ότι κάθε χρήστης έχει το δικό του ξεχωριστό προφίλ.
Η μέθοδος set() δημιουργεί ή αντικαθιστά το έγγραφο με τα επιθυμητά πεδία:

- displayName: περιέχει το όνομα που έχει δηλώσει ο χρήστης κατά την εγγραφή ή τροποποίηση του προφίλ του.
- createdAt: αποθηκεύεται με χρήση της εντολής FieldValue.serverTimestamp(), η οποία εξασφαλίζει ότι η χρονική σήμανση προέρχεται από τον server της Google και όχι από τη συσκευή του χρήστη, προσφέροντας ακρίβεια και ακεραιότητα στα χρονικά δεδομένα.

Η αποθήκευση προφίλ στο Firestore επιτρέπει στην εφαρμογή να δημιουργήσει επεκτάσιμη και δυναμική υποδομή διαχείρισης χρηστών, επιτρέποντας την ευέλικτη ανάκτηση, ενημέρωση και ανάλυση δεδομένων προφίλ για μελλοντική χρήση.

Κανόνες ασφαλείας Firestore (μόνο ο ίδιος ο χρήστης βλέπει/γράφει το έγγραφό του)

Για την προστασία των δεδομένων των χρηστών και την αποτροπή μη εξουσιοδοτημένης πρόσβασης, το Firebase παρέχει ένα ευέλικτο σύστημα κανόνων ασφαλείας (Firestore Security Rules), μέσω του οποίου μπορεί να καθοριστεί ποιοι χρήστες έχουν δικαίωμα ανάγνωσης ή εγγραφής σε συγκεκριμένα έγγραφα ή συλλογές.

Στην παρούσα υλοποίηση, κάθε χρήστης διατηρεί προσωπικό έγγραφο προφίλ εντός της συλλογής users. Για να διασφαλιστεί ότι μόνο ο ίδιος ο χρήστης έχει δικαίωμα ανάγνωσης και εγγραφής στο αντίστοιχο έγγραφό του, εφαρμόζεται ο ακόλουθος κανόνας ασφαλείας:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {

    // root-doc του χρήστη
    match /users/{userId} {
      allow read, write: if request.auth != null
                         && request.auth.uid == userId;
    }

    // κάθε υποσυλλογή του χρήστη (1 επίπεδο βάθος)
    match /users/{userId}/{collection}/{doc} {
      allow read, write: if request.auth != null
                         && request.auth.uid == userId;
    }

    // optional: read-only global fallback
    match /settings/global/app {
      allow read: if true;
    }
  }
}

```

Η παραπάνω δήλωση λειτουργεί ως εξής:

- Η εντολή match /users/{userId} ορίζει ότι ο κανόνας εφαρμόζεται σε όλα τα έγγραφα της συλλογής users, ανεξαρτήτως συγκεκριμένου userId.
- Η συνθήκη request.auth != null διασφαλίζει ότι η πρόσβαση επιτρέπεται μόνο σε πιστοποιημένους χρήστες (authenticated).
- Η συνθήκη request.auth.uid == userId ελέγχει ότι το UID του πιστοποιημένου χρήστη ταυτίζεται με το αναγνωριστικό του εγγράφου στο οποίο επιχειρεί πρόσβαση – γεγονός που προϋποθέτει ότι κατά την αποθήκευση κάθε προφίλ έχει χρησιμοποιηθεί ως όνομα εγγράφου το αντίστοιχο UID.

Με αυτόν τον τρόπο, η εφαρμογή εφαρμόζει έναν μηχανισμό αυτοπεριοριζόμενης πρόσβασης, κατά τον οποίο κάθε χρήστης έχει αποκλειστική πρόσβαση μόνο στα δικά του δεδομένα, ενώ τα δεδομένα άλλων χρηστών παραμένουν απομονωμένα και απρόσιτα. Αυτή η προσέγγιση ενισχύει την προστασία των προσωπικών δεδομένων, συμμορφώνεται με τις αρχές της ασφάλειας εξ ορισμού (security by design) και αποτελεί βέλτιστη πρακτική για εφαρμογές που διαχειρίζονται προφίλ ή ευαίσθητες πληροφορίες χρηστών.

Γ) Εξαγωγή Δεδομένων από Flutter & Firestore στο Google Drive

Διαδικασία Εξαγωγής με παραδείγματα

Βήμα 1: Ανάκτηση Δεδομένων από Firestore

```
Future<List<Map<String, dynamic>>> fetchFirestoreData(String collection) async {
  QuerySnapshot snapshot = await FirebaseFirestore.instance.collection(collection).get();
  return snapshot.docs.map((doc) => doc.data() as Map<String, dynamic>).toList();
}
```

Βήμα 2: Μετατροπή σε CSV/JSON

```
String convertToCsv(List<Map<String, dynamic>> data) {
  if (data.isEmpty) return '';
  final header = data.first.keys.join(',');
  final rows = data.map((e) => e.values.join(','));
  return '$header\n${rows.join('\n')}';
}
```

Βήμα 3: Αποθήκευση Προσωρινού Αρχείου

```
Future<File> saveTempFile(String content, String filename) async {
  final dir = await getTemporaryDirectory();
  final file = File('${dir.path}/$filename');
  await file.writeAsString(content);
  return file;
}
```

Βήμα 4: Ανεβάσμα στο Google Drive

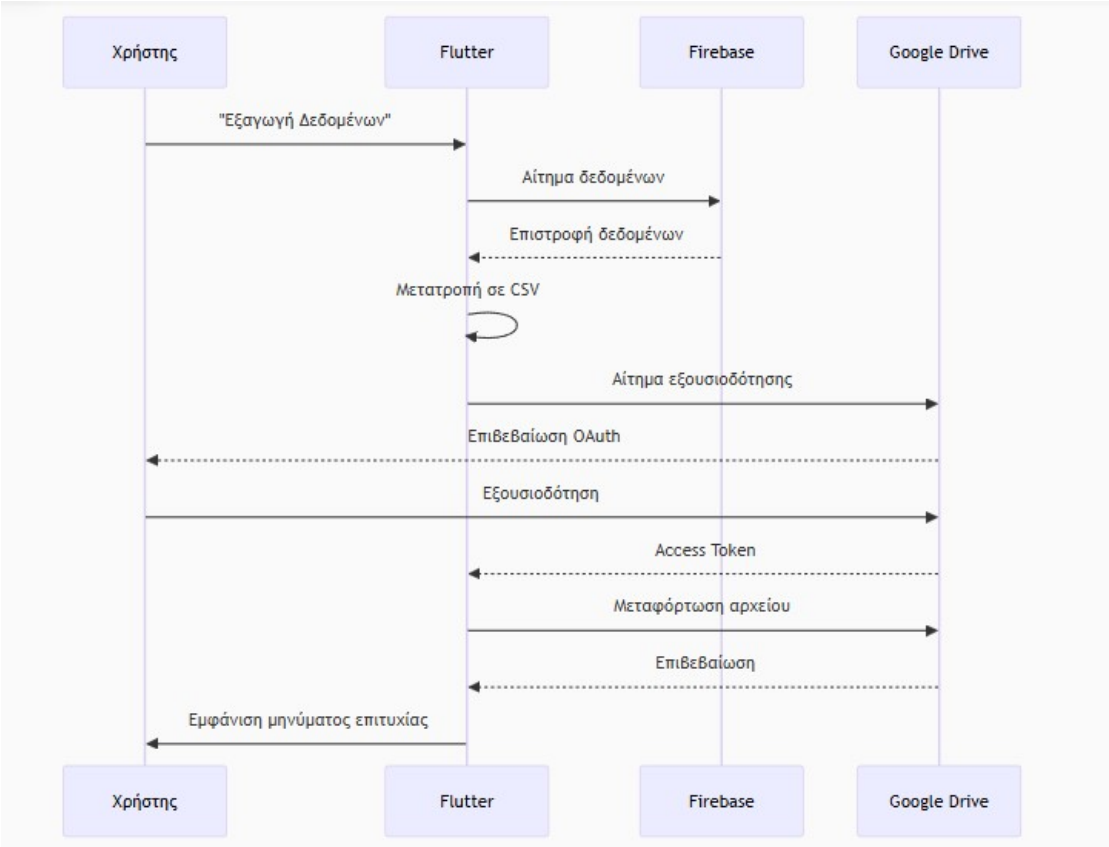
```
Future<void> uploadToDrive(File file, BuildContext context) async {
  final GoogleSignInAccount? account = await GoogleSignIn().signIn();
  final authHeaders = await account!.authHeaders;

  final driveApi = drive.DriveApi(
    GoogleHttpClient(authHeaders),
  );

  final media = drive.Media(file.openRead(), await file.length());
  await driveApi.files.create(
    drive.File()..name = file.uri.pathSegments.last,
    uploadMedia: media,
  );

  ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(content: Text('Επιτυχής εξαγωγή στο Google Drive!'))
  );
}
```

Σενάριο Χρήσης

1. Ο χρήστης επιλέγει "Εξαγωγή Δεδομένων" από το UI

2. Η εφαρμογή ζητά εξουσιοδότηση Google OAuth
3. Τα δεδομένα ανακτώνται από Firestore
4. Μετατρέπονται σε CSV/JSON
5. Το αρχείο αποθηκεύεται προσωρινά
6. Μεταφέρεται στο Drive του χρήστη

Διαγράμματα Ροής



Δ) Διαχείριση Κατάστασης με Riverpod

Εισαγωγή

Στην εφαρμογή Flutter που αναπτύχθηκε, υιοθετήθηκε το Riverpod ως βασικό πλαίσιο διαχείρισης κατάστασης. Η επιλογή αυτή βασίστηκε στην ανάγκη για:

- Ασφαλή και προβλέψιμη διαχείριση κατάστασης
- Αποτελεσματική διαχείριση ασύγχρονων λειτουργιών
- Ευκολία στη δοκιμασιμότητα και συντήρηση κώδικα
- Εξαρτηματοποιημένη αρχιτεκτονική

Υλοποίηση & Αρχιτεκτονική

Provider Hierarchy:

```
final apiServiceProvider = Provider<ApiService>((ref) => ApiService());

final userRepositoryProvider = Provider<UserRepository>((ref) {
  final apiService = ref.watch(apiServiceProvider);
  return UserRepository(apiService);
});

final authStateProvider = StateNotifierProvider<AuthNotifier, AuthState>((ref) {
  final userRepo = ref.watch(userRepositoryProvider);
  return AuthNotifier(userRepo);
});
```

Δομή εξαρτήσεων με κλιμακούμενη επίλυση (dependency injection)

Κατάσταση UI & Λογικής:

```
class AuthNotifier extends StateNotifier<AuthState> {
  final UserRepository _repo;

  AuthNotifier(this._repo) : super(AuthInitial());

  Future<void> login(String email, String password) async {
    state = AuthLoading();
    try {
      final user = await _repo.authenticate(email, password);
      state = AuthAuthenticated(user);
    } catch (e) {
      state = AuthError(e.toString());
    }
  }
}
```

Πρόσβαση σε Widgets:

```
class LoginScreen extends ConsumerWidget {
  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final authState = ref.watch(authStateProvider);

    return authState.when(
      initial: () => _buildLoginForm(ref),
      loading: () => CircularProgressIndicator(),
      authenticated: (user) => Dashboard(user),
      error: (err) => ErrorView(err),
    );
  }
}
```

Τεχνικά Πλεονεκτήματα

1. Ασφάλεια Τύπων:
    a. Compile-time έλεγχος για λανθασμένες αναφορές providers
    b. Αυτόματη ανίχνευση μη διαθέσιμων dependencies
2. Απόδοση:
    a. Πρότυπα auto-dispose για αυτόματη απελευθέρωση μνήμης
    b. Επιλεκτική ανακατασκευή widgets με ref.watch()

```
final filteredProducts = ref.watch(productsProvider
    .select((products) => products.where((p) => p.inStock)));
```

3. Testing:

```
testWidgets('Login success', (tester) async {
  await tester.pumpWidget(
    ProviderScope(
      overrides: [
        authStateProvider.overrideWithValue(AuthAuthenticated(testUser))
      ],
      child: MaterialApp(home: LoginScreen()),
    ),
  );
  expect(find.text('Welcome!'), findsOneWidget);
});
```

4. Διαχείριση Πλευρικών Επιδράσεων (Side Effects Management) :

```
ref.listen(authStateProvider, (previous, next) {
  if (next is AuthError) {
    showErrorDialog(context, next.error);
  }
});
```

απόσπασμα κώδικα που χειρίζεται σφάλματα επαλήθευσης (authentication errors) χρησιμοποιώντας το Riverpod.

Σύγκριση με Άλλα Πλαίσια

Χαρακτηριστικό	Riverpod	Provider	BLoC
Εξάρτηση Context	Όχι	Ναι	Όχι
Compile-Safe	✅	❌	✅
Auto-Dispose	✅	❌	⚠️ (Manual)
Testability	⭐⭐⭐⭐⭐	⭐⭐⭐⭐	⭐⭐⭐⭐
Learning Curve	Μέτριο	Εύκολο	Απαιτητικό

Προκλήσεις & Λύσεις

1. Πρόβλημα: Δυσκολία στην αρχικοποίηση πολύπλοκων dependencies

Λύση: Χρήση ProviderScope με overrides για integration testing

2. Πρόβλημα: Memory leaks σε μεγάλες ροές δεδομένων

Λύση: Εφαρμογή .autoDispose σε όλους τους providers με transient κατάσταση

```
final userSessionProvider = StateProvider.autoDispose((ref) => Session());
```

Ε) Διαχείριση Ραντεβού

Στοιχεία αρχείου & εισαγωγές

Στοιχείο	Περιγραφή
Όνομα αρχείου	**appointments_page.dart**
Σκοπός	Οθόνη «Διαχείριση ραντεβού» που επιτρέπει **προβολή, δημιουργία, επεξεργασία και διαγραφή** (CRUD) ραντεβού ανά χρήστη, με αποθήκευση στο Cloud Firestore.
Βιβλιοθήκες	• flutter/material.dart: Widgets/UI• cloud_firestore.dart: real-time βάση δεδομένων• firebase_auth.dart: ανάκτηση uid τρέχοντος χρήστη• mylib/mywidgets.dart: προσαρμοσμένα widgets (π.χ. BorderedBox)

2. Δομή σε υψηλό επίπεδο

```dart
class AppointmentsPage extends StatelessWidget {
  const AppointmentsPage({super.key});

  @override
  Widget build(BuildContext context) {
    final now = Timestamp.now();

    return Scaffold(
      appBar: AppBar(title: const Text('Διαχείριση ραντεβού')),
```

Η οθόνη υλοποιείται ως StatelessWidget· η αλληλεπίδραση γίνεται μέσω Streams και Dialogs, άρα δεν χρειάζεται stateful root. Όλα τα mutable δεδομένα (λίστα ραντεβού, φόρμες) ζουν:

1) Στο Firestore (remote state) μέσω StreamBuilder.
2) Τοπικά μέσα σε StatefulBuilder του διαλόγου.

3. Πρόσβαση στη συλλογή ραντεβού

```dart
// ---------------- Firestore ----------------
CollectionReference<Map<String, dynamic>> _appointmentsRef() {
  final uid = FirebaseAuth.instance.currentUser!.uid;
  return FirebaseFirestore.instance
      .collection('users')
      .doc(uid)
      .collection('appointments');
}
```

- «Ρίζα» paths: users/{uid}/appointments — κάθε χρήστης βλέπει μόνο τα δικά του έγγραφα.
- Η μέθοδος καλείται κάθε φορά που απαιτείται ref, αποφεύγοντας global variables.

4. Παρακολούθηση δεδομένων (Realtime UI)

```
body: StreamBuilder<QuerySnapshot<Map<String, dynamic>>>(
  stream:
      _appointmentsRef()
          .orderBy('dateTime')
          .where('dateTime', isGreaterThan: now)
          .limit(30)
          .snapshots(),
  builder: (context, snap) {
```

- orderBy('dateTime'): ταξινομεί χρονικά.
- where('dateTime', isGreaterThan: now): εμφανίζει μόνο μελλοντικά ραντεβού.
- limit(30): προστασία UI / quota.
- Η StreamBuilder ανανεώνει αυτόματα τη λίστα σε κάθε αλλαγή Firestore (real-time).

5. Προβολή λίστας & UI χειρισμοί

- Κενή λίστα → εμφανίζεται μήνυμα + κουμπί «Νέο ραντεβού».

- Μη κενή λίστα:
- BorderedBox για στυλ περίγραμματος.
- ListView.separated με ListTile ανά ραντεβού.
- Τίτλος = ημερομηνία + ώρα + τίτλος ραντεβού.
- IconButton** (✎)** → επεξεργασία
- IconButton** (🗑)** → doc.reference.delete()

```
final docs = snap.data!.docs;
if (docs.isEmpty) {
  return Center(
    child: Column(
      mainAxisSize: MainAxisSize.min,
      children: [
        const Text('Δεν υπάρχουν ραντεβού'),
        const SizedBox(height: 8),
        ElevatedButton.icon(
          icon: const Icon(Icons.add),
          label: const Text('Νέο ραντεβού'),
          onPressed: () => _showAppointmentDialog(context),
        ), // ElevatedButton.icon
      ],
    ), // Column
  ); // Center
}
```

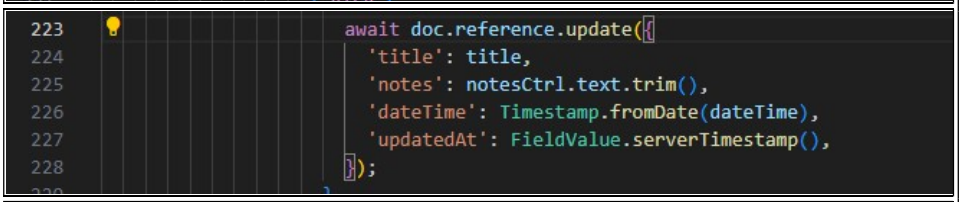
```
return BorderedBox(
  child: ListView.separated(
    itemCount: docs.length,
    separatorBuilder: (_, __) => const Divider(height: 0),
    itemBuilder: (context, i) {
      final doc = docs[i];
      final data = doc.data();
      final dt = (data['dateTime'] as Timestamp).toDate();

      return ListTile(
        title: Text(
          '${dt.day}/${dt.month}/${dt.year}  '
          '${dt.hour.toString().padLeft(2, "0")}:'
          '${dt.minute.toString().padLeft(2, "0")}\n'
          '${data['title'] ?? ''}',
          style: const TextStyle(fontWeight: FontWeight.bold),
        ), // Text
        subtitle: Text('${data['notes'] ?? ''}'),
        trailing: Row(
          mainAxisSize: MainAxisSize.min,
          children: [
            // ---------- EDIT ----------
            IconButton(
              icon: const Icon(Icons.edit, size: 20),
              tooltip: 'Επεξεργασία',
              onPressed:
                  () => _showAppointmentDialog(context, doc: doc),
            ), // IconButton
            // ---------- DELETE ----------
            IconButton(
              icon: const Icon(Icons.delete, size: 20),
              tooltip: 'Διαγραφή',
              onPressed: () => doc.reference.delete(),
            ), // IconButton
          ],
        ), // Row
      ); // ListTile
    },
  ), // ListView.separated
); // BorderedBox
```

6. Διάλογος Προσθήκης / Επεξεργασίας

```
void _showAppointmentDialog(
  BuildContext context, {
  DocumentSnapshot<Map<String, dynamic>>? doc,
}) {
  final data = doc?.data();
  final titleCtrl = TextEditingController(text: data?['title'] ?? '');
  final notesCtrl = TextEditingController(text: data?['notes'] ?? '');
  DateTime? selectedDateTime =
      data?['dateTime'] != null
          ? (data!['dateTime'] as Timestamp).toDate()
          : null;

  showDialog(
    context: context,
    builder:
        (ctx) => StatefulBuilder(
          builder: (ctx, setState) {
            Future<void> pickDateTime() async {
              final now = DateTime.now();
              // ημερομηνία
              final date = await showDatePicker(
                context: ctx,
                useRootNavigator: false,
                firstDate: now,
                initialDate: selectedDateTime ?? now,
                lastDate: DateTime(now.year + 5),
              );
              if (date == null || !ctx.mounted) return;
```

- void _showAppointmentDialog(BuildContext context, {DocumentSnapshot? doc})
- Αν doc == null → λειτουργία ADD, αλλιώς EDIT.
- StatefulBuilder χρησιμοποιείται για local state (selectedDateTime).
- Βήματα επιλογής ημερομηνίας/ώρας:
- showDatePicker() – περιορισμένο από σήμερα έως +5 χρόνια.
- showTimePicker() – ώρα.
- Validation: Αν τίτλος ή ημερομηνία λείπουν εμφανίζεται SnackBar.

7. Εντολές Firestore

Λειτουρ.	Κώδικας
ADD	
EDIT	
DELETE	

Ζ) Δημιουργία Εγγράφων

1. Στοιχεία αρχείου & εισαγωγές

Στοιχείο	Περιγραφή
Όνομα αρχείου	**create_doc.dart**
Σκοπός	Οθόνη «Έγγραφα» που επιτρέπει προβολή, δημιουργία, επεξεργασία, εξαγωγή και αποθήκευση υπηρεσιακών εγγράφων ανά χρήστη, με σύνδεση στο **Firestore** και το **Google Drive.**
Βιβλιοθήκες	• `flutter/material.dart`: UI και φόρμες • `cloud_firestore.dart`: real-time αποθήκευση και ανάκτηση • `firebase_auth.dart`: ανάκτηση `uid` χρήστη • `googleapis` / `http`: αποστολή στο Google Drive • `mylib/mywidgets.dart`: επαναχρησιμοποιούμενα widgets • `mylib/myfunctions.dart`: βοηθητικές συναρτήσεις (formatting, validation) • `drive_service.dart`: διασύνδεση με Google Drive API

2. Πίνακας Μεθόδων

Όνομα Μεθόδου	Είσοδος	Έξοδος	Λειτουργία
`addDoc()`	Τιμές από φόρμα	Νέο έγγραφο στο Firestore	Δημιουργεί νέα εγγραφή με timestamps και τιμές από πεδία φόρμας.
`updateDoc()`	Τιμές από φόρμα και `docRef`	Τροποποιημένο έγγραφο	Ενημερώνει υπάρχον έγγραφο με νέες τιμές και ενημερωμένο timestamp.
`deleteDoc(DocumentReference)`	Αναφορά εγγράφου	Διαγραφή	Αφαιρεί μόνιμα το επιλεγμένο έγγραφο από το Firestore.
`exportDoc(doc)`	DocumentSnapsh	Δημιουργία	Εξάγει το έγγραφο στο Google Docs μέσω Apps

	ot	Google Doc	Script, στέλνοντας τα πεδία.
`showDialog()`	`BuildContext`, έγγραφο (προαιρετικά)	Popup εισαγωγής	Παρουσιάζει τη φόρμα δημιουργίας/επεξεργασίας για είσοδο/τροποποίηση.
`showDatePicker()`	Context και ημερομηνία	DateTime	Παρέχει στον χρήστη δυνατότητα επιλογής ημερομηνιών.
`_snack(String)`	Μήνυμα string	Εμφάνιση στο UI	Προβάλλει σύντομο SnackBar μήνυμα ενημέρωσης στο κάτω μέρος της οθόνης.
`_populateForm(Map)`	Map με δεδομένα εγγράφου	Ενημέρωση πεδίων	Γεμίζει όλα τα πεδία φόρμας με δεδομένα για προβολή ή επεξεργασία.
`_clearForm()`	—	Εκκαθάριση φόρμας	Καθαρίζει όλα τα πεδία φόρμας.
`_pickDateGeneric(...)`	Τρέχουσα ημερομηνία και setter function	Επιλεγμένη ημερομηνία	Ανοίγει picker για να επιλεγεί ημερομηνία και την καταχωρεί.
`_saveDoc()`	—	Firestore Add/Update	Αποθηκεύει (νέο ή ενημερωμένο) έγγραφο στο Firestore.
`_deleteDoc(String)`	Αναγνωριστικό εγγράφου	Διαγραφή Firestore	Διαγράφει έγγραφο από τη συλλογή εγγράφων του χρήστη.
`_exportDoc(Map)`	Δεδομένα εγγράφου	Google Docs output	Εξάγει έγγραφο μέσω GAS με template.

3. Δομή σε υψηλό επίπεδο

Η αρχιτεκτονική της οθόνης βασίζεται σε μία κύρια Stateful κλάση (TemplateDropdownPage), η οποία οργανώνει τα επιμέρους widgets και διαχειρίζεται την κατάσταση της εφαρμογής. Το UI περιλαμβάνει ένα DropdownButtonFormField για επιλογή τύπου εγγράφου (π.χ. άδεια, υπερωρία), το οποίο επηρεάζει άμεσα το StreamBuilder που παρακολουθεί και εμφανίζει δυναμικά τα αποθηκευμένα έγγραφα του επιλεγμένου τύπου από τη βάση Firestore.

Παράλληλα, η φόρμα εγγράφου είναι ενσωματωμένη μέσα στην ίδια σελίδα και ελέγχεται με Form και GlobalKey<FormState>, ενώ τα δεδομένα που συμπληρώνει ο χρήστης είτε προστίθενται ως νέα entries είτε ενημερώνουν υπάρχουσες εγγραφές. Όλες οι λειτουργίες (read, write, delete, export) υλοποιούνται μέσω ξεχωριστών βοηθητικών συναρτήσεων ή εξωτερικών service αρχείων (drive_service.dart). Η εξαγωγή βασίζεται σε integration με το Google Drive και Google Docs API, απομονώνοντας τη λογική σύνδεσης στο cloud.

Η απόκριση του UI παραμένει real-time χάρη στο stream από Firestore και η επεκτασιμότητα είναι διασφαλισμένη μέσω δυναμικού εντοπισμού templateId και πεδίων, χωρίς ανάγκη τροποποίησης του κώδικα για νέους τύπους εγγράφων.

```
407   @override
408   Widget build(BuildContext context) {
409     final templatesAsync = ref.watch(templatesProvider);
410     final selectedTemplateId = ref.watch(selectedTemplateIdProvider);
411     final uid = ref.watch(authUidProvider);
412
413     return Scaffold(
414       appBar: AppBar(title: const Text('Έγγραφα')),
415       floatingActionButton: FloatingActionButton(
416         onPressed: () {
417           if (selectedTemplateId == null) {
418             _snack('Πρώτα επιλέξτε template');
419             return;
420           }
421           setState(() {
422             _showForm = true;
423             _viewMode = false;
424             _editing = false;
425             _editingDocId = null;
426           });
427           _clearForm();
428         },
429         child: const Icon(Icons.add),
430       ),
431       body: Padding(
432         padding: const EdgeInsets.all(16),
433         child: Column(
434           crossAxisAlignment: CrossAxisAlignment.stretch,
435           children: [
436             // dropdown
```

3. Πρόσβαση σε έγγραφα

Η εφαρμογή αποκτά πρόσβαση στα έγγραφα του κάθε χρήστη μέσω διαδρομής Firestore της μορφής:

users/{uid}/templates/{templateId}/docs

Η διαδρομή βασίζεται στο αναγνωριστικό του χρήστη (uid) που παρέχεται από το Firebase Auth, και στον τύπο προτύπου (templateId) που επιλέγεται δυναμικά από το

dropdown. Έτσι, εξασφαλίζεται απομόνωση των δεδομένων ανά χρήστη και ταξινόμηση ανά κατηγορία εγγράφου.

H) GOOGLE SCRIPTS

Α) Απαραίτητα για τη δημιουργία Εγγράφων από Πρότυπα DOC του google drive.

```
// ---------------------------------------------------------------------------
// Drive Template Service (με ανίχνευση και fallback για placeholders)
// ---------------------------------------------------------------------------
const DEFAULT_FOLDER_ID = '1gQCEPMZ5y4PJX9NW73qQHMejfHN-FPcL';

function json(obj) {
  return ContentService
    .createTextOutput(JSON.stringify(obj))
    .setMimeType(ContentService.MimeType.JSON);
}

function doGet(e) {
  try {
    const action = (e.parameter.action || 'listTemplates').trim();
    switch (action) {
      case 'listTemplates':
        return listTemplates(e.parameter.folderId || DEFAULT_FOLDER_ID);
      case 'listKeys':
        return listTemplateKeys(e.parameter.templateId);
      default:
        return json({ ok: false, message: `Unknown action "${action}"` });
    }
  } catch (err) {
    console.error(err);
    return json({ ok: false, message: err.message, stack: err.stack });
  }
}

function doPost(e) {
  try {
    const data = JSON.parse(e.postData.contents || '{}');
    const { folderId, templateId, filename } = data;
    if (!folderId || !templateId || !filename) {
      return json({ ok: false, message: 'Missing folderId | templateId | filename' });
    }
    const out = createFromTemplate(data);
    return json({ ok: true, ...out });
  } catch (err) {
    console.error(err);
    return json({ ok: false, message: err.message, stack: err.stack });
  }
}

function listTemplates(folderId) {
  try {
    const list = [];
    const files = DriveApp.getFolderById(folderId).getFiles();
    while (files.hasNext()) {
      const f = files.next();
      list.push({ name: f.getName(), fileId: f.getId() });
    }
    list.sort((a, b) => a.name.toLowerCase().localeCompare(b.name.toLowerCase()));
    return json({ ok: true, templates: list });
  } catch (err) {
    console.error(err);
    return json({ ok: false, message: `Cannot read folder ${folderId}`, stack: err.stack });
  }
}
```

```
function createFromTemplate(data) {
  const { folderId, templateId, filename, record = {} } = data;
  const copy = DriveApp.getFileById(templateId).makeCopy(filename, DriveApp.getFolderById(folderId));
  const doc = DocumentApp.openById(copy.getId());
  replacePlaceholders(doc, record);
  doc.saveAndClose();

  return {
    fileId: copy.getId(),
    url: copy.getUrl(),
    filename,
    timestamp: new Date().toISOString(),
  };
}
```

```
function listTemplateKeys(templateId) {
  try {

    console.log('templateId:', templateId);


    const doc = DocumentApp.openById(templateId);
    const parts = [doc.getBody(), doc.getHeader(), doc.getFooter()].filter(Boolean);
    const keys = new Set();

    parts.forEach(p => {
      const matches = p.getText().match(/{{[^{}]+}}/g);
      if (matches) {
        matches.forEach(ph => {
          const key = ph.replace(/[{}]/g, '').trim();
          keys.add(key);
        });
      }
    });

    return json({ ok: true, keys: [...keys] });
  } catch (err) {
    console.error(err);
    return json({ ok: false, message: `Cannot read template: ${err.message}`, stack: err.stack });
  }
}


// ---------------------------------------------------------------------------
// Αντικατάσταση placeholders με fallback "ΑΓΝΩΣΤΟ:<όνομα>"
// ---------------------------------------------------------------------------
function replacePlaceholders(doc, record) {
  const parts = [doc.getBody(), doc.getHeader(), doc.getFooter()].filter(Boolean);
  const foundPlaceholders = new Set();

  // Βρες όλα τα {{placeholders}} στο έγγραφο
  parts.forEach(p => {
    const matches = p.getText().match(/{{[^{}]+}}/g);
    if (matches) matches.forEach(ph => foundPlaceholders.add(ph));
  });

  const replacements = {};

  foundPlaceholders.forEach(ph => {
    const key = ph.replace(/[{}]/g, '').trim();
    replacements[ph] = key in record ? record[key] : `ΑΓΝΩΣΤΟ:${key}`;
  });

  // Εκτέλεση αντικαταστάσεων
  Object.entries(replacements).forEach(([ph, val]) => {
    parts.forEach(p => p.replaceText(ph, val));
  });
}
```

Β)) Απαραίτητα για τη δημιουργία Ημερολόγιο Συμβάντων στο google drive.

```
function doPost(e) {
  try {
    // 1. Parse JSON payload
    const payload  = JSON.parse(e.postData.contents);
    const folderId = payload.folderId;
    const uid      = payload.uid;
    const records  = payload.records;

    // 2. Βρες ή δημιούργησε το Spreadsheet
    const folder   = DriveApp.getFolderById(folderId);
    const fileName = 'Βιβλιο Συμβάντων'  ;
    const ss       = getOrCreateSpreadsheetInFolder_(folder, fileName);

    // 3. Καθάρισμα & append rows
    const sheet = ss.getSheets()[0];
    sheet.clearContents();
    sheet.appendRow(['Ημερομηνία', 'Συμβάν']);
    records.forEach(r => sheet.appendRow([r.date, r.event]));

    // 4. Επιστροφή επιτυχούς αποτελέσματος
    return ContentService
      .createTextOutput('OK')
      .setMimeType(ContentService.MimeType.TEXT);
  } catch (err) {
    // Σε σφάλμα, επιστρέφουμε 200 + μήνυμα σφάλματος
    return ContentService
      .createTextOutput('ERROR: ' + err.toString())
      .setMimeType(ContentService.MimeType.TEXT);
  }
}

/**
 * Βοηθητική: βρίσκει ή δημιουργεί φάκελο "old" και μετακινεί παλιό αρχείο
 * με timestamp.
 */
function getOrCreateSpreadsheetInFolder_(folder, name) {
  const files = folder.getFilesByName(name);
  if (files.hasNext()) {
    const existingFile = files.next();
    const oldFolder    = getOrCreateSubfolder_(folder, 'old');

    const now = new Date();
    const ts  = Utilities.formatDate(now, Session.getScriptTimeZone(), 'yyyy-MM-dd_HH-mm');
    existingFile.setName(name + '_' + ts);

    oldFolder.addFile(existingFile);
    folder.removeFile(existingFile);
  }

  const ss      = SpreadsheetApp.create(name);
  const newFile = DriveApp.getFileById(ss.getId());
  folder.addFile(newFile);
  return ss;
}

function getOrCreateSubfolder_(parentFolder, name) {
  const folders = parentFolder.getFoldersByName(name);
  return folders.hasNext() ? folders.next() : parentFolder.createFolder(name);
}

```

Βιβλιογραφία

1. Google Flutter Team. (2024). *Flutter Documentation*. Διαθέσιμο στο: https://docs.flutter.dev123
2. Google Firebase Team. (2024). *Firebase Documentation*. Διαθέσιμο στο: https://firebase.google.com/docs45
3. Google Developers. (2024). *Apps Script Guides*. Διαθέσιμο στο: https://developers.google.com/apps-script6
4. Rousselet, R. (2023). *Riverpod Official Documentation*. Διαθέσιμο στο: https://riverpod.dev7
5. Code With Andrea. (2024). *Flutter Riverpod 2.0: The Ultimate Guide*. Διαθέσιμο στο: https://codewithandrea.com/articles/flutter-state-management-riverpod/7
6. Flutter Team. (2025). *Flutter Learning Resources*. Διαθέσιμο στο: https://docs.flutter.dev/reference/learning-resources8
7. Firebase Open Source Docs. (2024). *GitHub - firebase/firebase-docs*. Διαθέσιμο στο: https://github.com/firebase/firebase-docs