ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή Εργασία

Τίτλος Πτυχιακής Εργασίας	Μελέτη και Υλοποίηση Android Εφαρμογής Διαχείρισης Προσωπικών Εξόδων με Δυνατότητα OCR Αποδείξεων Study and Implementation of Android Expense Management Application with Receipt OCR Capabilities
Ονοματεπώνυμο Φοιτητή	Διονύσιος Παναγιώτης Χριστοδουλόπουλος
Πατρώνυμο	Κωνσταντίνος
Αριθμός Μητρώου	Π22200
Επιβλέπων	Ευθύμιος Αλέπης, Καθηγητής

Ημερομηνία Παράδοσης Ιούλιος 2026

Copyright ©

Ευχαριστίες

Με την ολοκλήρωση της παρούσας πτυχιακής εργασίας, θα ήθελα να ευχαριστήσω θερμά όλους όσοι συνέβαλαν στην πραγματοποίησή της. Αρχικά, ευχαριστώ τον επιβλέποντα καθηγητή μου, κ. Ευθύμιο Αλέπη, για την εμπιστοσύνη και την υποστήριξη που μου προσέφερε καθ' όλη τη διάρκεια εκπόνησης της εργασίας. Θα ήθελα, επίσης, να εκφράσω την ευγνωμοσύνη μου στην οικογένειά μου για την υπομονή, την κατανόηση και τη συνεχή στήριξή της σε κάθε στάδιο των σπουδών μου. Τέλος, ευχαριστώ όλους όσοι, με οποιονδήποτε τρόπο, συνέβαλαν και με στήριξαν στην ακαδημαϊκή και προσωπική μου πορεία.

Περίληψη

Η παρούσα πτυχιακή εργασία παρουσιάζει τη μελέτη, τον σχεδιασμό, την ανάπτυξη και την αξιολόγηση μιας εφαρμογής Android για τη διαχείριση προσωπικών εξόδων, η οποία αξιοποιεί τεχνολογία Οπτικής Αναγνώρισης Χαρακτήρων (Optical Character Recognition - OCR) με σκοπό την αυτοματοποίηση της καταχώρισης οικονομικών συναλλαγών. Βασικός στόχος της εφαρμογής είναι η απλοποίηση της διαδικασίας καταγραφής εξόδων, επιτρέποντας στον χρήστη να σαρώνει αποδείξεις αγορών μέσω της κάμερας της κινητής συσκευής και να εξάγει αυτόματα βασικά στοιχεία, όπως η επωνυμία του καταστήματος, το συνολικό ποσό της συναλλαγής και η κατηγορία της δαπάνης.

Η εφαρμογή αναπτύχθηκε αξιοποιώντας σύγχρονες τεχνολογίες και αρχιτεκτονικές ανάπτυξης εφαρμογών Android, όπως οι Kotlin, Jetpack Compose, Clean Architecture, MVVM, Room Database, CameraX και Dagger Hilt. Για την αναγνώριση του κειμένου ενσωματώθηκε το Google Cloud Vision API, προσφέροντας υψηλή ακρίβεια στην επεξεργασία αποδείξεων, ιδιαίτερα στην ελληνική γλώσσα. Τα εξαγόμενα δεδομένα αποθηκεύονται τοπικά στη συσκευή, επιτρέποντας στον χρήστη να οργανώνει τις δαπάνες του, να παρακολουθεί προϋπολογισμούς, να αναλύει τις καταναλωτικές του συνήθειες μέσω στατιστικών διαγραμμάτων και να εξάγει τα δεδομένα σε μορφή CSV.

Η υλοποίηση δίνει έμφαση στη συντηρησιμότητα, την επεκτασιμότητα, την ευχρηστία και την προστασία των προσωπικών δεδομένων, εφαρμόζοντας σύγχρονες πρακτικές μηχανικής λογισμικού και τοπική αποθήκευση δεδομένων. Η αξιολόγηση της εφαρμογής έδειξε ότι η χρήση τεχνολογίας OCR μειώνει σημαντικά την ανάγκη χειροκίνητης καταχώρισης, διατηρώντας ικανοποιητικό επίπεδο ακρίβειας σε πραγματικές συνθήκες χρήσης. Το τελικό σύστημα αποτελεί μια ολοκληρωμένη και πρακτική λύση για την προσωπική οικονομική διαχείριση, ενώ παράλληλα δημιουργεί τις προϋποθέσεις για μελλοντικές επεκτάσεις, όπως η αξιοποίηση τεχνητής νοημοσύνης για αυτόματη κατηγοριοποίηση, ο συγχρονισμός μέσω υπολογιστικού νέφους και η παροχή προηγμένων εργαλείων οικονομικής ανάλυσης.

Λέξεις Κλειδιά: *Android, Οπτική Αναγνώριση Χαρακτήρων (OCR), Google Cloud Vision API, Kotlin, Jetpack Compose, Καθαρή Αρχιτεκτονική (Clean Architecture), MVVM, CameraX, Room Database, Ευρετικοί Αλγόριθμοι, Σάρωση Αποδείξεων, Διαχείριση Προσωπικών Οικονομικών, Material Design 3, Dagger Hilt*

Abstract

This thesis presents the design, development, and evaluation of an Android application for personal expense management that incorporates Optical Character Recognition (OCR) technology to automate the recording of financial transactions. The primary objective of the application is to simplify the expense tracking process by allowing users to scan purchase receipts using their mobile device's camera and automatically extract essential information, such as the store name, transaction amount, and expense category.

The application was developed using modern Android technologies and architectural principles, including Kotlin, Jetpack Compose, Clean Architecture, MVVM, Room Database, CameraX, and Dagger Hilt. For text recognition, Google's Cloud Vision API was integrated to provide accurate OCR capabilities, particularly for Greek-language receipts. The extracted data are stored locally, enabling users to organize expenses, monitor budgets, visualize spending patterns through statistical charts, and export financial records in CSV format.

The implementation emphasizes maintainability, scalability, usability, and data privacy by adopting contemporary software engineering practices and local data storage. The evaluation of the application demonstrates that OCR significantly reduces manual data entry while maintaining satisfactory recognition accuracy under typical usage conditions. The resulting system offers a practical and efficient solution for personal financial management while providing a solid foundation for future enhancements, including AI-assisted categorization, cloud synchronization, and advanced financial analytics.

Key Words: ***Android, Optical Character Recognition (OCR), Google Cloud Vision API, Kotlin, Jetpack Compose, Clean Architecture, MVVM, CameraX, Room Database, Heuristic Algorithms, Receipt Scanning, Personal Finance Management, Material Design 3, Dagger Hilt***

Πίνακας Περιεχομένων

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Κατάλογος Διαγραμμάτων

1 Εισαγωγή

Η ταχεία εξέλιξη των κινητών συσκευών έχει αλλάξει ριζικά τον τρόπο με τον οποίο διαχειριζόμαστε την καθημερινότητά μας, από την επικοινωνία μέχρι την οργάνωση των οικονομικών μας. Η δυνατότητα άμεσης πρόσβασης σε ισχυρούς επεξεργαστές, υψηλής ποιότητας κάμερες και υπηρεσίες υπολογιστικού νέφους (cloud computing) επιτρέπει την ανάπτυξη "έξυπνων" εφαρμογών που αυτοματοποιούν χρονοβόρες διαδικασίες. Μία από αυτές τις διαδικασίες είναι η καταγραφή και διαχείριση των προσωπικών εξόδων. Παραδοσιακά, η διαχείριση εξόδων απαιτεί χειροκίνητη πληκτρολόγηση ποσών και ημερομηνιών, κάτι που συχνά οδηγεί σε παραλείψεις και λάθη.

Η παρούσα πτυχιακή εργασία εστιάζει στο σχεδιασμό και την ανάπτυξη μιας σύγχρονης εφαρμογής Android για τη διαχείριση εξόδων (“Expense Tracker”), η οποία ενσωματώνει τεχνολογίες Οπτικής Αναγνώρισης Χαρακτήρων (Optical Character Recognition - OCR). Η ενσωμάτωση OCR επιτρέπει στους χρήστες να σαρώνουν φυσικές αποδείξεις αγορών μέσω της κάμερας του κινητού τους και να εξάγουν αυτόματα τα απαραίτητα δεδομένα, όπως το συνολικό ποσό πληρωμής, την επωνυμία του καταστήματος καθώς και την κατηγορία στην οποία ενδεχομένως ανήκει.

1.1 Αντικείμενο και Σκοπός της Πτυχιακής

Το κύριο αντικείμενο της πτυχιακής εργασίας είναι η υλοποίηση ενός ολοκληρωμένου συστήματος σε περιβάλλον Android που λειτουργεί ως έξυπνος ψηφιακός βοηθός οικονομικών.

Ο σκοπός της εργασίας είναι διττός: Αφενός, η εξοικείωση και η εφαρμογή σύγχρονων προτύπων σχεδιασμού λογισμικού, όπως η αρχιτεκτονική "Clean Architecture" και το πρότυπο MVVM (Model-View-ViewModel), καθώς και η χρήση σύγχρονων εργαλείων δημιουργίας διεπαφών χρήστη, όπως το Jetpack Compose. Αφετέρου, η μελέτη, ενσωμάτωση και παραμετροποίηση αλγορίθμων μηχανικής όρασης (Cloud Vision API, CameraX) για την αξιόπιστη αναγνώριση και επεξεργασία κειμένου από φυσικές αποδείξεις στα Ελληνικά, ένα πρόβλημα που παρουσιάζει σημαντικές δυσκολίες λόγω της ποικιλομορφίας των αποδείξεων (γραμματοσειρές, ποιότητα χαρτιού, σκιές, διάταξη).

1.2 Κίνητρο και Χρησιμότητα της Εφαρμογής

Το κίνητρο για την ανάπτυξη της συγκεκριμένης εφαρμογής πηγάζει από την ανάγκη για γρηγορότερη και πιο αξιόπιστη καταγραφή των καθημερινών εξόδων. Οι περισσότερες δωρεάν εφαρμογές στο Google Play Store απαιτούν από το χρήστη να εισάγει με το χέρι (manual entry) κάθε έξοδο ή καταγράφουν αυτόματα μόνο έξοδα που έχουν ολοκληρωθεί με πιστωτική/χρεωστική κάρτα. Αν και η διαδικασία αυτή είναι απλή, όταν ο όγκος των αποδείξεων μεγαλώνει (π.χ. μετά από επίσκεψη σε σούπερ μάρκετ ή σε ταξίδια), η χειροκίνητη καταχώρηση γίνεται κουραστική.

Η εφαρμογή μας στοχεύει να επιλύσει αυτό το πρόβλημα συνδυάζοντας:

- **Ταχύτητα:** Με ένα "κλικ" της κάμερας ανακτώνται τα δεδομένα.
- **Ακρίβεια:** Μειώνονται τα ανθρώπινα λάθη (typos) κατά την πληκτρολόγηση αριθμών.
- **Οργάνωση:** Παρέχεται οπτικοποίηση των δεδομένων μέσω διαγραμμάτων (πίτα, ρα-βδογράμματα) και κατάλληλη κατηγοριοποίηση για την καλύτερη κατανόηση των καταναλωτικών συνηθειών.

1.3 Δομή της Εργασίας

Η παρούσα πτυχιακή εργασία είναι οργανωμένη με τέτοιο τρόπο ώστε να οδηγήσει τον αναγνώστη ομαλά από τις βασικές θεωρητικές έννοιες, στο σχεδιασμό και, τελικά, στην πρακτική υλοποίηση και αξιολόγηση του συστήματος. Η διάρθρωση των κεφαλαίων έχει ως εξής:

- Στο **Κεφάλαιο 2** παρουσιάζεται το θεωρητικό υπόβαθρο του οικοσυστήματος Android, η γλώσσα προγραμματισμού Kotlin και οι σύγχρονες αρχιτεκτονικές λογισμικού (Clean Architecture, MVVM). Επίσης, αναλύονται τα εργαλεία Material Design, CameraX και OCR.
- Στο **Κεφάλαιο 3** γίνεται η ανάλυση των απαιτήσεων του συστήματος και παρατίθενται τα διαγράμματα περιπτώσεων χρήσης (Use Cases), καθώς και ο σχεδιασμός της τοπικής βάσης δεδομένων.
- Το **Κεφάλαιο 4** αποτελεί τον πυρήνα της υλοποίησης όσον αφορά την αρχιτεκτονική. Εξετάζεται η κατασκευή της εφαρμογής επίπεδο προς επίπεδο (Data, Domain, Presentation) και ο τρόπος υλοποίησης των διεπαφών με το Jetpack Compose.
- Στο **Κεφάλαιο 5** αξιολογείται η εφαρμογή, τόσο ως προς την ακρίβεια του OCR όσο και ως προς την εμπειρία χρήστη, ενώ καταγράφονται προβλήματα που προέκυψαν.
- Τέλος, στο **Κεφάλαιο 6** συνοψίζονται τα συμπεράσματα, οι περιορισμοί της εφαρμογής και προτείνονται πιθανές μελλοντικές επεκτάσεις του συστήματος.

2 Θεωρητικό Υπόβαθρο και Τεχνολογίες

Στο κεφάλαιο αυτό παρουσιάζεται το θεωρητικό υπόβαθρο των τεχνολογιών και των εργαλείων λογισμικού που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής. Η κατανόηση αυτών των εννοιών είναι κρίσιμη για την ανάλυση της αρχιτεκτονικής του συστήματος στα επόμενα κεφάλαια. Η επιλογή των συγκεκριμένων εργαλείων βασίστηκε στις πλέον σύγχρονες πρακτικές ανάπτυξης που προτείνει η Google για το οικοσύστημα του Android.

2.1 Το Λειτουργικό Σύστημα Android & η Γλώσσα Kotlin

Το Android αποτελεί το πιο διαδεδομένο λειτουργικό σύστημα για κινητές συσκευές παγκοσμίως. Είναι βασισμένο σε έναν τροποποιημένο πυρήνα του Linux (Linux kernel) και παρέχει ένα ανοιχτό περιβάλλον (open-source) για την ανάπτυξη εφαρμογών (Sills, 2023). Παραδοσιακά, η ανάπτυξη εφαρμογών για το Android γινόταν σχεδόν αποκλειστικά σε γλώσσα Java.

Εικόνα 2.1 Λογότυπο του Android

Ωστόσο, τα τελευταία χρόνια, η Google ανακήρυξε την Kotlin ως την επίσημη και προτεινόμενη γλώσσα για την ανάπτυξη Android. Η Kotlin είναι μια σύγχρονη, στατικά τυποποιημένη (statically typed) γλώσσα προγραμματισμού, η οποία προσφέρει πλήρη διαλειτουργικότητα (interoperability) με τη Java (JetBrains, 2025; Elizarov, 2024). Τα βασικά αρχιτεκτονικά και λειτουργικά πλεονεκτήματα που δικαιολογούν την επιλογή της για την υλοποίηση της παρούσας εφαρμογής συνοψίζονται στα εξής:

- **Ασφάλεια Τύπων και Διαχείριση Κενών Δεδομένων (Null-Safety):** Το σύστημα τύπων της Kotlin διαχωρίζει ρητά τις αναφορές που μπορούν να λάβουν τιμή null από αυτές που δεν μπορούν. Με τον τρόπο αυτό, μεταφέρει τον εντοπισμό πιθανών σφαλμάτων τύπου NullPointerException (NPE) από το χρόνο εκτέλεσης (runtime) στο χρόνο μεταγλώττισης (compile-time), διασφαλίζοντας τη σταθερότητα και την αξιοπιστία της εφαρμογής.
- **Συντακτική Συνοχή και Μείωση Επαναλαμβανόμενου Κώδικα (Conciseness):** Η γλώσσα περιορίζει δραματικά την ανάγκη συγγραφής επαναλαμβανόμενου κώδικα (boilerplate code) μέσω δυνατοτήτων όπως τα data classes, τα extension functions και το type inference. Το γεγονός αυτό μειώνει την πιθανότητα λογικών σφαλμάτων και καθιστά τον κώδικα πιο ευανάγνωστο και εύκολα συντηρήσιμο.
- **Αποδοτικός Ασύγχρονος Προγραμματισμός (Coroutines & Flows):** Για τη διαχείριση ενεργοβόρων διεργασιών στο παρασκήνιο, όπως η επικοινωνία με το δίκτυο ή η προσπέλαση της τοπικής βάσης δεδομένων, η Kotlin εισάγει τις ρουτίνες συντονισμού (Coroutines) και τις ροές δεδομένων (Flows). Το μοντέλο αυτό επιτρέπει τη συγγραφή ασύγχρονου κώδικα με σειριακή/γραμμική δομή, εξασφαλίζοντας ότι το κύριο νήμα της διεπαφής (UI Main Thread) παραμένει ανεμπόδιστο (non-blocking) για το χρήστη.
- **Εγγενής Υποστήριξη Σύγχρονων Εργαλείων UI (Jetpack Compose):** Η Kotlin αποτελεί το θεμέλιο για το Jetpack Compose, το σύγχρονο, δηλωτικό (declarative) toolkit της Google για την ανάπτυξη διεπαφών χρήστη. Η χρήση του Compose επιτρέπει την πλήρη περιγραφή του UI μέσω κώδικα Kotlin (καταργώντας τα παραδοσιακά αρχεία XML), γεγονός που επιταχύνει τη διαδικασία ανάπτυξης, διευκολύνει τη δυναμική ανανέωση της οθόνης βάσει της κατάστασης (state) της εφαρμογής, και δένει αρμονικά με την αρχιτεκτονική MVVM.

Εικόνα 2.2 Λογότυπο της Kotlin

2.2 Σύγχρονες Αρχιτεκτονικές Λογισμικού

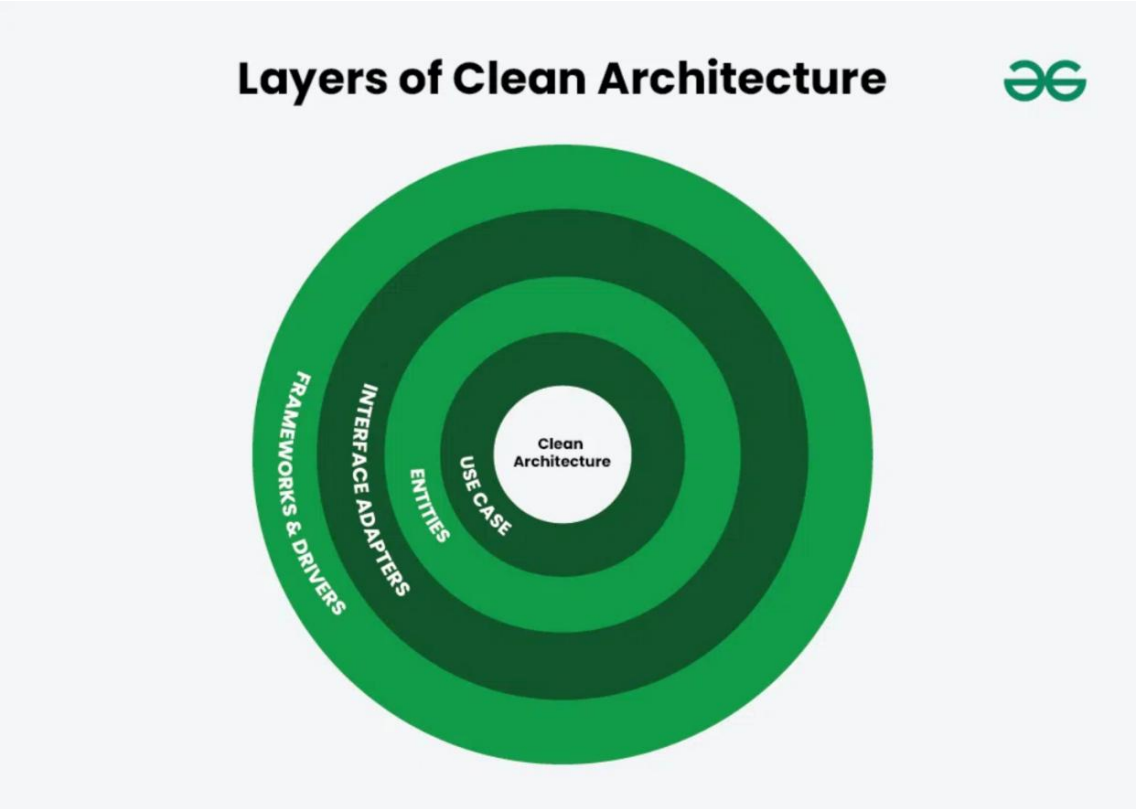
Η ανάπτυξη μιας εφαρμογής χωρίς τη χρήση συγκεκριμένης αρχιτεκτονικής προσέγγισης οδηγεί αναπόφευκτα σε στενά συνδεδεμένο κώδικα (tightly coupled). Σε τέτοιες περιπτώσεις, η διεπαφή χρήστη (UI), η επιχειρησιακή λογική και η πρόσβαση στα δεδομένα συστεγάζονται στο ίδιο επίπεδο (π.χ. εντός μιας κλάσης Activity ή Fragment). Η πρακτική αυτή καθιστά τον κώδικα δυσλειτουργικό, μη ελέγξιμο μέσω αυτοματοποιημένων δοκιμών (untestable) και εξαιρετικά δυσχερή στη συντήρηση ή την επέκτασή του. Για την αντιμετώπιση αυτών των προκλήσεων, στην παρούσα εργασία υιοθετήθηκε ένας συνδυασμός δύο καθιερωμένων αρχιτεκτονικών προτύπων (Android Developers, 2025).

2.2.1 Καθαρή Αρχιτεκτονική (Clean Architecture)

Η Καθαρή Αρχιτεκτονική (Clean Architecture), η οποία προτάθηκε από τον Robert C. Martin, στοχεύει στο διαχωρισμό των ευθυνών του συστήματος (separation of concerns) σε ομόκεντρα επίπεδα (layers). Ο θεμελιώδης κανόνας της αρχιτεκτονικής αυτής είναι ο Κανόνας των Εξαρτήσεων (Dependency Rule), ο οποίος ορίζει ότι οι πηγαίες εξαρτήσεις πρέπει να κατευθύνονται μόνο προς τα μέσα. Αυτό σημαίνει ότι ο εσωτερικός πυρήνας δεν διατηρεί καμία γνώση για τις τεχνικές λεπτομέρειες των εξωτερικών επιπέδων, όπως η βάση δεδομένων ή το framework του Android (GeeksforGeeks, 2024; Martin, 2018).

Στο πλαίσιο της εφαρμογής, η αρχιτεκτονική δομείται σε τρία βασικά επίπεδα:

- **Domain Layer (Εσωτερικό Επίπεδο):** Αποτελεί την καρδιά της εφαρμογής και περιέχει την καθαρή επιχειρησιακή λογική (Business Logic). Περιλαμβάνει τις περιπτώσεις χρήσης (Use Cases) και τις οντότητες (Models). Είναι πλήρως απομονωμένο και ανεξάρτητο από βιβλιοθήκες τρίτων και από το ίδιο το Android framework.
- **Data Layer (Εξωτερικό Επίπεδο):** Είναι υπεύθυνο για τη διαχείριση και την άντληση των δεδομένων, είτε από τοπικές πηγές (π.χ. βάση δεδομένων SQLite/Room) είτε από απομακρυσμένες (π.χ. REST APIs για την επεξεργασία OCR). Στο επίπεδο αυτό υλοποιείται το πρότυπο Repository Pattern, το οποίο λειτουργεί ως διαμεσολαβητής, αποκρύπτοντας την πηγή προέλευσης των δεδομένων από το Domain Layer.
- **Presentation Layer (Εξωτερικό Επίπεδο):** Είναι υπεύθυνο για τη σχεδίαση και τη συμπεριφορά της διεπαφής χρήστη (UI), καθώς και για την παρουσίαση των δεδομένων που επεξεργάζεται το Domain Layer.

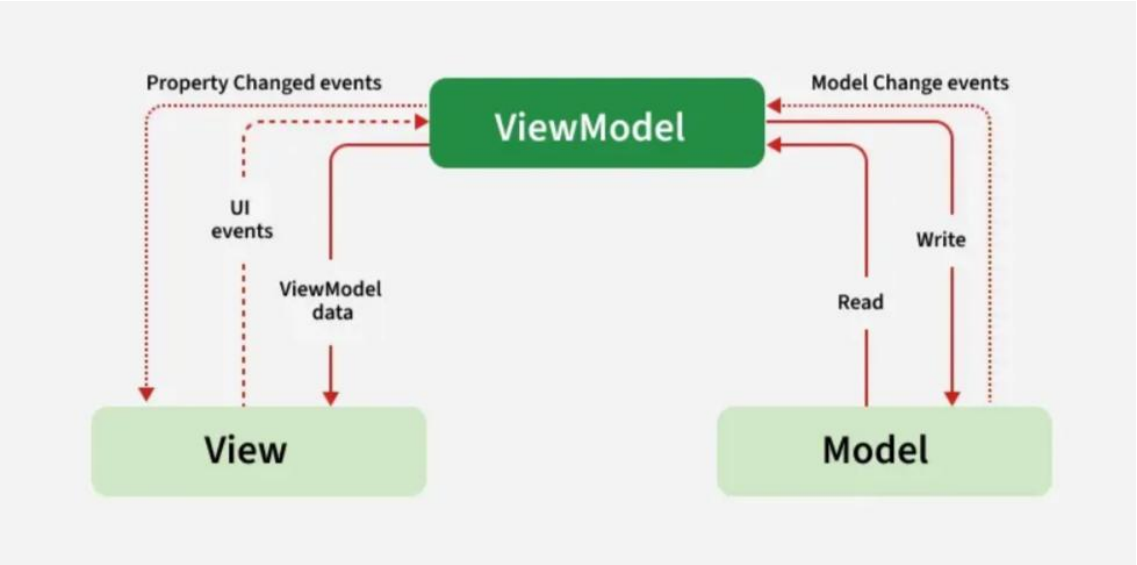


Εικόνα 2.3 Τα επίπεδα της Clean Architecture

2.2.2 Το Πρότυπο MVVM (Model-View-ViewModel)

Εντός του Presentation Layer, για την αποτελεσματική διαχείριση της ροής δεδομένων και της κατάστασης της οθόνης, εφαρμόζεται το αρχιτεκτονικό πρότυπο MVVM (Model-View-ViewModel), το οποίο αποτελεί την επίσημη σύσταση της Google για το Android (GeeksforGeeks, 2024):

- **Model:** Αντιπροσωπεύει το στρώμα των δεδομένων και της επιχειρησιακής λογικής. Στο περιβάλλον της Clean Architecture, το Model του MVVM ουσιαστικά ταυτίζεται και επικοινωνεί απευθείας με τα Use Cases του Domain Layer.
- **View:** Αφορά αποκλειστικά το οπτικό μέρος της εφαρμογής. Είναι υπεύθυνο για την απεικόνιση των δεδομένων στην οθόνη και τη μεταβίβαση των αλληλεπιδράσεων του χρήστη (clicks, inputs) στο ViewModel, χωρίς να εκτελεί καμία λογική επεξεργασία.
- **ViewModel:** Λειτουργεί ως συνδετικός κρίκος μεταξύ του View και του Model. Συγκρατεί και εκθέτει την τρέχουσα κατάσταση της οθόνης (UI State). Το βασικό αρχιτεκτονικό του πλεονέκτημα στο Android είναι ότι είναι σχεδιασμένο να επιβιώνει από αλλαγές διαμόρφωσης της συσκευής (configuration changes), όπως η περιστροφή της οθόνης, αποτρέποντας την απώλεια δεδομένων και την άσκοπη επανεκκίνηση των διεργασιών. Η επικοινωνία του με το View επιτυγχάνεται ασύγχρονα, μέσω παρατηρήσιμων ροών δεδομένων (StateFlow/SharedFlow), διασφαλίζοντας μια αντιδραστική (reactive) αρχιτεκτονική.



Εικόνα 2.4 Σχέδιο του MVVM

2.3 Έγχυση Εξαρτήσεων (Dependency Injection) με το Dagger Hilt

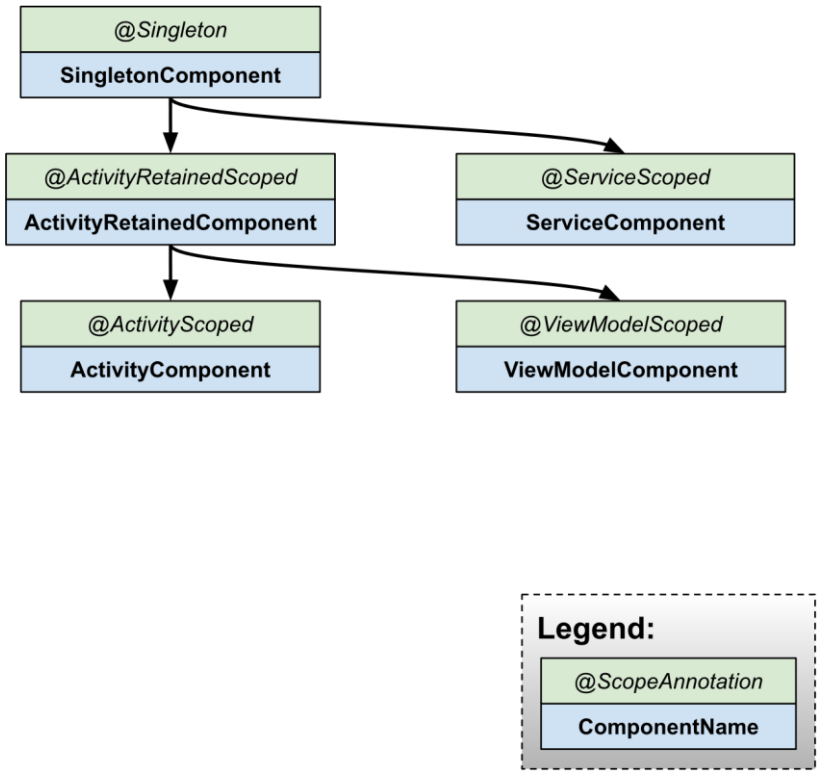
Η Έγχυση Εξαρτήσεων (Dependency Injection - DI) αποτελεί ένα θεμελιώδες σχεδιαστικό πρότυπο (design pattern) που υλοποιεί την αρχή της Αντιστροφής Ελέγχου (Inversion of Control - IoC). Σύμφωνα με το πρότυπο αυτό, μια κλάση (π.χ. ένα ViewModel) δεν αναλαμβάνει την ευθύνη για τη δημιουργία και την αρχικοποίηση των αντικειμένων από τα οποία εξαρτάται (π.χ. ένα Repository). Αντίθετα, οι απαραίτητες εξαρτήσεις τροφοδοτούνται ("εγχέονται") έτοιμες στην κλάση, συνήθως μέσω του κατασκευαστή της (constructor injection), από μια εξωτερική, κεντρική οντότητα, η οποία ονομάζεται διαχειριστής εξαρτήσεων (DI Container).

Για την υλοποίηση αυτής της αρχιτεκτονικής στην παρούσα εφαρμογή επιλέχθηκε το Dagger Hilt. Το Hilt αποτελεί ένα επίπεδο αφαίρεσης (abstraction layer) αναπτυγμένο πάνω στο ισχυρό αλλά πολύπλοκο framework Dagger 2, με σκοπό την απλοποίηση της ενσωμάτωσής του σε περιβάλλον Android (Android Developers, 2025; Dagger, n.d.).

Το Hilt παρέχει προκαθορισμένα "δοχεία εξαρτήσεων" (Hilt Components), τα οποία είναι άρρηκτα συνδεδεμένα με τον κύκλο ζωής των βασικών στοιχείων του Android framework. Ενδεικτικά:

- Το **SingletonComponent** χρησιμοποιείται για τη διαχείριση αντικειμένων των οποίων η διάρκεια ζωής ταυτίζεται με αυτή της ίδιας της εφαρμογής (π.χ. η βάση δεδομένων ή η κλάση διαχείρισης δικτύου).
- Το **ViewModelComponent** περιορίζει τη διάρκεια ζωής των εξαρτήσεων αποκλειστικά στον κύκλο ζωής του αντίστοιχου ViewModel.

Η δήλωση και η συσχέτιση των εξαρτήσεων πραγματοποιείται μέσω μετα-δεδομένων (Annotations), όπως τα @Inject, @Module και @InstallIn. Το Hilt επεξεργάζεται αυτά τα annotations κατά το χρόνο μεταγλώττισης (compile-time code generation), γεγονός που εξασφαλίζει τη βέλτιστη δυνατή απόδοση (καθώς αποφεύγεται η δαπανηρή χρήση του reflection κατά το runtime) και εγγυάται τη δημιουργία ενός αρθρωτού (modular) και εύκολα ελέγξιμου κώδικα μέσω unit tests.



Εικόνα 2.5 Ιεραρχία Component του Hilt

2.4 Σχεδιασμός Διεπαφής με το Jetpack Compose

Παραδοσιακά, η κατασκευή γραφικών περιβαλλόντων στο Android βασιζόταν σε αρχεία διάταξης μορφής XML, ακολουθώντας το επιτακτικό πρότυπο σχεδιασμού (Imperative UI). Στο μοντέλο αυτό, η διεπαφή χρήστη διατηρούσε μια δική της, ξεχωριστή κατάσταση από τη λογική της εφαρμογής. Ως εκ τούτου, ο προγραμματιστής ήταν υπεύθυνος για την αναζήτηση των οπτικών στοιχείων μέσω της μεθόδου findViewById() και τη χειροκίνητη ενημέρωση της κατάστασής τους (π.χ. μέσω της μεθόδου setText()) ως απόκριση σε διάφορα συμβάντα (events). Η προσέγγιση αυτή αύξανε την πολυπλοκότητα, καθώς ο χειροκίνητος συγχρονισμός μεταξύ των δεδομένων και της οθόνης οδηγούσε συχνά σε σφάλματα ασυγχρονισμού (state desynchronization).

Για την αντιμετώπιση αυτών των περιορισμών, στην παρούσα εφαρμογή χρησιμοποιήθηκε το Jetpack Compose, το σύγχρονο, εγγενές (native) εργαλείο της Google για την ανάπτυξη δηλωτικού γραφικού περιβάλλοντος (Declarative UI) (Android Developers, 2025). Αντί για τη χρήση αρχείων XML, η διεπαφή χρήστη αναπτύσσεται εξ ολοκλήρου σε περιβάλλον Kotlin μέσω ειδικών συναρτήσεων, γνωστών ως @Composable functions.

Στο δηλωτικό αυτό παράδειγμα, ο προγραμματιστής περιγράφει ("δηλώνει") πως πρέπει να προβάλλεται η οθόνη για μια δεδομένη κατάσταση δεδομένων (UI State). Όταν η κατάσταση αυτή μεταβάλλεται (π.χ. κατά την εισαγωγή νέων δεδομένων από το ViewModel), το framework του Jetpack Compose εκτελεί αυτόματα τη διαδικασία της ανασύνθεσης (Recomposition). Κατά τη διαδικασία αυτή, το σύστημα επανυπολογίζει και σχεδιάζει εκ νέου, με ιδιαίτερα αποδοτικό τρόπο, αποκλειστικά και μόνο τα τμήματα της οθόνης που

επηρεάζονται από τη συγκεκριμένη αλλαγή (Google Developers, 2024). Η αρχιτεκτονική αυτή εξαλείφει την ανάγκη χειροκίνητου συγχρονισμού, ελαχιστοποιεί τις πιθανότητες εμφάνισης λογικών σφαλμάτων, διασφαλίζει την οπτική συνοχή της εφαρμογής και επιταχύνει σημαντικά τη διαδικασία ανάπτυξης (rapid UI development).

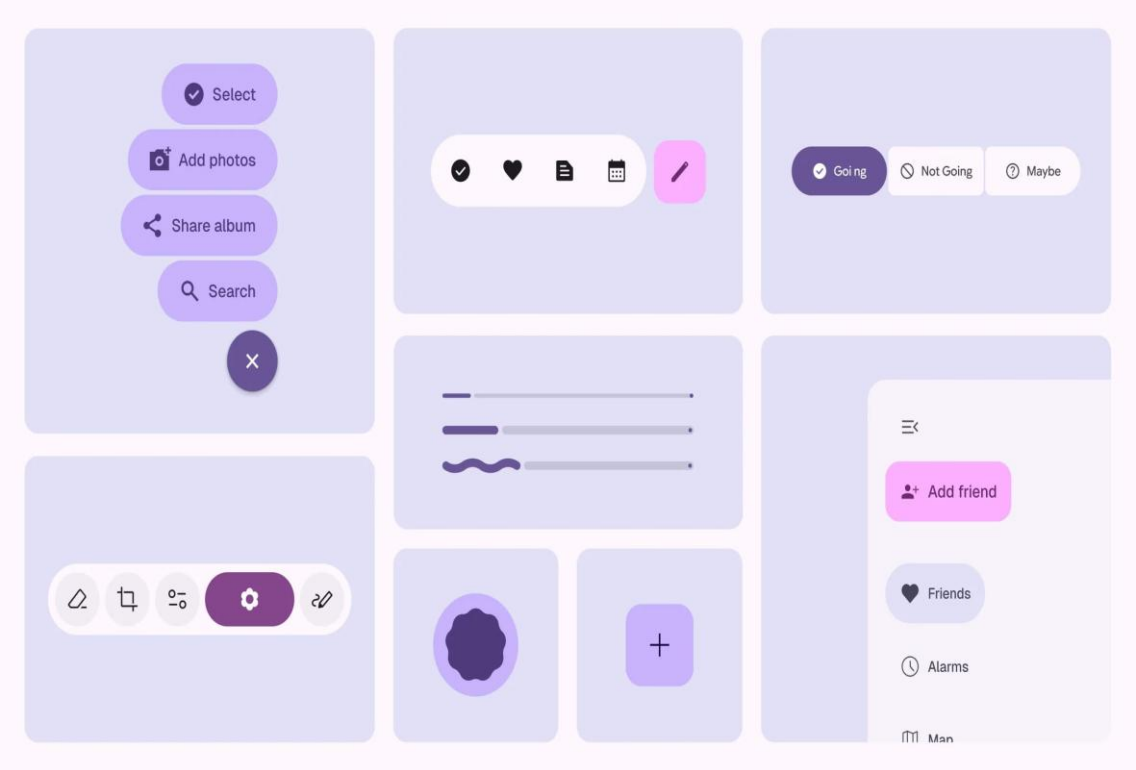
2.5 Το Σχεδιαστικό Σύστημα Material Design 3

Το Material Design αποτελεί το επίσημο σύστημα σχεδίασης (design system) που αναπτύχθηκε από τη Google, με σκοπό την ενοποίηση της εμπειρίας χρήστη (User Experience - UX) και τη διασφάλιση της οπτικής συνοχής σε όλα τα προϊόντα και τις πλατφόρμες λογισμικού. Στην παρούσα εφαρμογή αξιοποιήθηκε η νεότερη έκδοσή του, το Material Design 3 (γνωστό και ως Material You), το οποίο ενσωματώνεται εγγενώς και επεκτείνει τις δυνατότητες του Jetpack Compose.

Τα βασικά αρχιτεκτονικά και λειτουργικά χαρακτηριστικά του Material Design 3 που διευκόλυναν την ανάπτυξη και τη βελτιστοποίηση της διεπαφής συνοψίζονται στα εξής:

- **Συστατικά Στοιχεία Διεπαφής (M3 Components):** Το πρότυπο παρέχει μια ευρεία γκάμα έτοιμων, παραμετροποιήσιμων δομικών στοιχείων (π.χ. Floating Action Buttons, Material Cards, Navigation Bars και Dialogs). Τα στοιχεία αυτά ενσωματώνουν βιομηχανικά πρότυπα εργονομίας και εγγυώνται τη συμβατότητα με τις διεθνείς οδηγίες προσβασιμότητας (accessibility), όπως η σωστή χρωματική αντίθεση και τα ελάχιστα μεγέθη περιοχών αφής.
- **Δυναμικός Χρωματισμός (Dynamic Color) και Design Tokens:** Το Material 3 εισάγει την αρχιτεκτονική των design tokens, η οποία επιτρέπει την αφαίρεση των τιμών σχεδίασης (χρώματα, σκιές) σε μορφή δυναμικών μεταβλητών. Μέσω αυτών, παρέχεται η δυνατότητα αυτόματης προσαρμογής της χρωματικής παλέτας της εφαρμογής βάσει των προτιμήσεων του χρήστη ή του υπόβαθρου (wallpaper) της συσκευής του, προσφέροντας μια εξαιρετικά προσωποποιημένη εμπειρία.
- **Σημασιολογική Χρωματική Θεωρία:** Το σύστημα ορίζει ένα αυστηρό χρωματικό σχήμα (Color Scheme) με σημασιολογικά χρώματα (π.χ. error, tertiary, success). Αυτά αξιοποιήθηκαν για την άμεση και διαισθητική κατανόηση της κατάστασης των οικονομικών δεδομένων από τον χρήστη (π.χ. διακριτή χρωματική σήμανση ανά κατηγορία δαπάνης ή προειδοποιήσεις υπέρβασης προϋπολογισμού).
- **Δομημένη Τυπογραφία (Typography Hierarchy):** Προσφέρεται μια σαφώς καθορισμένη ιεραρχία γραμματοσειρών κατηγοριοποιημένη σε Display, Headline, Title, Body, Label. Η δομή αυτή καθοδηγεί αποτελεσματικά την οπτική προσοχή του χρήστη στα κρισιμότερα δεδομένα της οθόνης.
- **Εγγενής Υποστήριξη Σκοτεινού Περιβάλλοντος (Dark Theme):** Το σύστημα παρέχει ενσωματωμένη αρχιτεκτονική για την άμεση μετάβαση σε σκοτεινή λειτουργία. Η υλοποίηση αυτή μειώνει την κούραση των ματιών σε συνθήκες χαμηλού φωτισμού και συμβάλλει στη βελτιστοποίηση της ενεργειακής απόδοσης της συσκευής, περιορίζοντας την κατανάλωση ενέργειας σε οθόνες τεχνολογίας OLED/AMOLED.

Η υιοθέτηση του Material Design 3 διασφαλίζει ότι η εφαρμογή “Expense Tracker” συμμορφώνεται πλήρως με τα σύγχρονα βιομηχανικά πρότυπα σχεδιασμού, παρουσιάζοντας τη συμπεριφορά μιας επαγγελματικής, εγγενούς (native) εφαρμογής Android, ενώ παράλληλα ελαχιστοποιεί την ανάγκη για σχεδιασμό γραφικών στοιχείων εξ αρχής.



Εικόνα 2.6 Material Design Στοιχεία

2.6 Τοπική Αποθήκευση Δεδομένων - Room Database

Στην ανάπτυξη εφαρμογών για κινητές συσκευές, η υποστήριξη της λειτουργίας εκτός σύνδεσης (offline support) αποτελεί κρίσιμη αρχιτεκτονική απαίτηση για τη διασφάλιση της απρόσκοπτης εμπειρίας του χρήστη. Στο περιβάλλον του Android, ο παραδοσιακός μηχανισμός για την αποθήκευση σχεσιακών δεδομένων βασίζεται στην ενσωματωμένη μηχανή βάσης δεδομένων SQLite. Ωστόσο, η απευθείας αλληλεπίδραση με την SQLite μέσω του χαμηλού επιπέδου κώδικα (low-level API) του Android ενέχει σημαντικά μειονεκτήματα, καθώς απαιτεί τη συγγραφή ερωτημάτων SQL υπό τη μορφή συμβολοσειρών (plain text strings). Η πρακτική αυτή καθιστά αδύνατο τον έλεγχο ορθότητας των ερωτημάτων κατά τη μεταγλώττιση, αυξάνοντας την πιθανότητα εμφάνισης σφαλμάτων κατά το χρόνο εκτέλεσης (runtime errors).

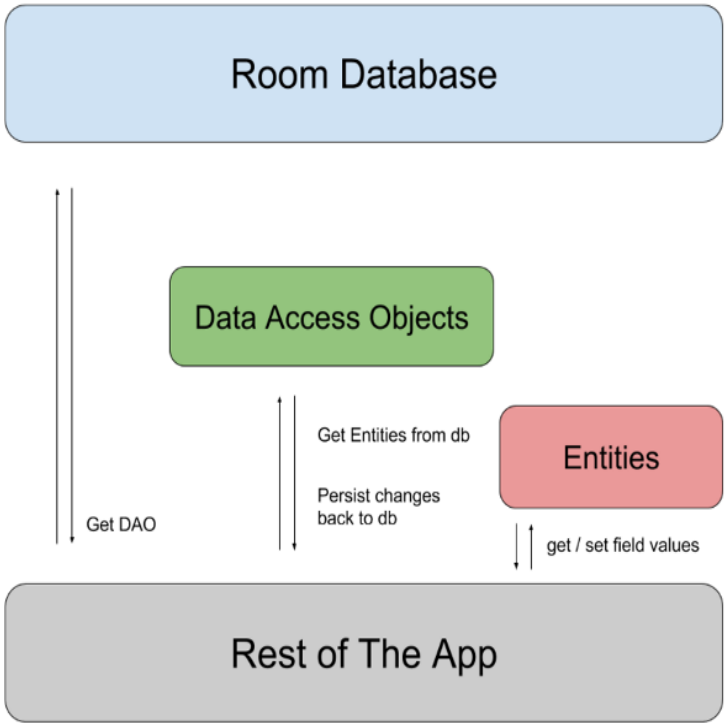
Για την αντιμετώπιση αυτών των περιορισμών, στην παρούσα εφαρμογή χρησιμοποιήθηκε η βιβλιοθήκη Room, η οποία ανήκει στο σύνολο βιβλιοθηκών Android Jetpack (Android Developers, 2025). Το Room λειτουργεί ως ένα επίπεδο αφαίρεσης (abstraction layer) πάνω από την SQLite, παρέχοντας δυνατότητες Αντικειμενοστρεφούς-Σχεσιακής Χαρτογράφησης (Object-Relational Mapping - ORM). Η αρχιτεκτονική της Room βασίζεται σε τρία κύρια συστατικά στοιχεία:

- **Entity (Οντότητα):** Αναπαριστά έναν πίνακα της σχεσιακής βάσης δεδομένων. Κάθε κλάση δεδομένων (data class) που επισημαίνεται ως οντότητα αντιστοιχίζεται αυτόματα σε έναν πίνακα της βάσης, όπου τα πεδία της κλάσης αποτελούν τις στήλες του πίνακα (για παράδειγμα, η οντότητα που αναπαριστά τα έξοδα/δαπάνες).
- **DAO (Data Access Object - Αντικείμενο Πρόσβασης Δεδομένων):** Αποτελεί μια διασύνδεση (interface) που ορίζει τις μεθόδους για την εκτέλεση λειτουργιών CRUD (Create, Read, Update, Delete) στη βάση δεδομένων. Η συσχέτιση των μεθόδων με τα αντίστοιχα ερωτήματα πραγματοποιείται μέσω Annotations

(όπως @Query, @Insert, @Delete). Το βασικό πλεονέκτημα του DAO είναι ότι το Room επικυρώνει τη συντακτική ορθότητα των ερωτημάτων SQL κατά το χρόνο μεταγλώττισης (compile-time verification), αποτρέποντας την εκτέλεση ελαττωματικού κώδικα.

- **Database (Βάση Δεδομένων):** Αποτελεί την κύρια κλάση που χρησιμεύει ως το κεντρικό σημείο πρόσβασης και σύνδεσης με την υποκείμενη SQLite βάση, διατηρώντας τη διαμόρφωση (configuration) και την τρέχουσα έκδοση (versioning) του σχήματος δεδομένων.

Επιπλέον, το Room ενσωματώνεται πλήρως με τις ασύγχρονες ροές δεδομένων της Kotlin (Kotlin Flows). Η δυνατότητα αυτή επιτρέπει την υιοθέτηση ενός αντιδραστικού μοντέλου (reactive model), όπου η εφαρμογή παρατηρεί (observes) τη βάση δεδομένων σε πραγματικό χρόνο. Κατά συνέπεια, οποιαδήποτε μεταβολή στα δεδομένα (π.χ. η καταχώριση μιας νέας δαπάνης μετά την επεξεργασία OCR μιας απόδειξης) πυροδοτεί την αυτόματη και δυναμική ενημέρωση της διεπαφής χρήστη (π.χ. την ανανέωση των στατιστικών γραφημάτων της αρχικής οθόνης), χωρίς να απαιτείται χειροκίνητη επαναφόρτωση από τον χρήστη.



Εικόνα 2.7 Διάγραμμα της αρχιτεκτονικής Room

2.7 Τεχνολογίες Μηχανικής Όρασης & AI (CameraX & Cloud Vision)

Ο πυρήνας αυτοματοποίησης της προτεινόμενης εφαρμογής βασίζεται στην ικανότητα ψηφιακής ανάγνωσης και εξαγωγής δεδομένων από φυσικές αποδείξεις λιανικής πώλησης. Η σύνθετη αυτή διεργασία απαιτεί τη συνδρομή συστημάτων προηγμένης λήψης εικόνας και τεχνολογιών οπτικής αναγνώρισης κειμένου, οι οποίες αναλύονται παρακάτω.

2.7.1 Η Βιβλιοθήκη CameraX

Για την υλοποίηση του υποσυστήματος λήψης φωτογραφιών χρησιμοποιήθηκε το CameraX, ένα σύγχρονο API της Google που ανήκει στο οικοσύστημα του Android Jetpack (Android Developers, 2025). Το CameraX σχεδιάστηκε για να γεφυρώσει τις αποκλίσεις και να αντιμετωπίσει τον έντονο κατακερματισμό (hardware fragmentation) των καμερών που χαρακτηρίζει τις συσκευές Android, εξασφαλίζοντας ενιαία και σταθερή συμπεριφορά ανεξαρτήτως κατασκευαστή.

Σε αντίθεση με παλαιότερες και πιο δύσκαμπτες υλοποιήσεις (όπως το Camera2 API), το CameraX διαθέτει εγγενή επίγνωση του κύκλου ζωής των στοιχείων της εφαρμογής (Lifecycle-Aware). Αυτό σημαίνει ότι το API διαχειρίζεται αυτόνομα και με ασφάλεια τη δέσμευση και την αποδέσμευση των πόρων της κάμερας, σε απόλυτο συντονισμό με την κατάσταση του αντίστοιχου Activity ή Fragment. Με τον τρόπο αυτό, ελαχιστοποιείται η άσκοπη κατανάλωση ενέργειας και εξαλείφεται η πιθανότητα εμφάνισης σφαλμάτων διαρροής μνήμης (memory leaks).

2.7.2 Οπτική Αναγνώριση Χαρακτήρων (OCR)

Η Οπτική Αναγνώριση Χαρακτήρων (Optical Character Recognition - OCR) αποτελεί τη διεργασία μετατροπής εικόνων που περιέχουν τυπωμένο, δακτυλογραφημένο ή χειρόγραφο κείμενο σε κωδικοποιημένη ψηφιακή πληροφορία, αναγνώσιμη και επεξεργάσιμη από υπολογιστικά συστήματα. Τα σύγχρονα συστήματα OCR βασίζονται σε αρχιτεκτονικές Βαθέων Νευρωνικών Δικτύων (Deep Neural Networks), τα οποία εκτελούν χωρική ανάλυση της εικόνας για τον εντοπισμό και την ταξινόμηση μορφολογικών προτύπων και χαρακτήρων.

Στο πλαίσιο της παρούσας εργασίας, οι αποδείξεις λιανικής συνιστούν μια ιδιαίτερα απαιτητική κατηγορία ανάλυσης εγγράφων (Document Analysis). Χαρακτηρίζονται ως "θορυβώδη" δεδομένα εισόδου, καθώς συχνά παρουσιάζουν παραμορφώσεις λόγω αναδίπλωσης του χαρτιού, χαμηλή αντίθεση εξαιτίας ξεθωριασμένου μελανιού, μη τυποποιημένες γραμματοσειρές και πολύπλοκη στοίχιση σε πολλαπλές στήλες.

Πίνακας 2.1 Συγκριτική Ανάλυση Μηχανών Οπτικής Αναγνώρισης Χαρακτήρων (OCR)

Κριτήριο Σύγκρισης	Google ML Kit	Tesseract OCR	Google Cloud Vision API
Τύπος Εκτέλεσης	On-Device (Τοπικά στη συσκευή)	On-Device (Τοπικά στη συσκευή)	Cloud-Based (Στο υπολογιστικό νέφος)
Απαίτηση Διαδικτύου	Όχι	Όχι	Ναι (Ενεργή σύνδεση)
Ανοικτού Κώδικα (Open Source)	Όχι (Proprietary SDK)	Ναι (Άδεια Apache 2.0)	Όχι (Proprietary Cloud API)
Υποστήριξη Ελληνικών	Περιορισμένη (χωρίς ειδικό ελληνικό OCR μοντέλο)	Ναι	Ναι (Βελτιστοποιημένη)
Ακρίβεια σε Αποδείξεις	Μέτρια προς χαμηλή	Μέτρια (εξαρτάται από προεπεξεργασία)	Υψηλή (Κλάσεις ανώτερη αναγνώριση χαρακτήρων)
Επίγνωση Διάταξης	Βασική (Blocks, γραμμές, στοιχεία κειμένου)	Περιορισμένη (Απαιτεί πολύπλοκη προεπεξεργασία)	Εξαιρετική (Ιεραρχική δομή: blocks, παραγράφους, λέξεις)
Κόστος Χρήσης	Δωρεάν	Δωρεάν	Freemium (δωρεάν όριο και χρέωση ανά χρήση)

2.7.3 Google Cloud Vision API

Δεδομένου ότι η επεξεργασία και η αναγνώριση ελληνικού κειμένου σε περιβάλλον αποδείξεων αποτελεί μια υπολογιστικά εντατική διεργασία που απαιτεί μοντέλα υψηλής ακρίβειας, επιλέχθηκε η προσέγγιση της αναγνώρισης στο υπολογιστικό νέφος (Cloud-based OCR), έναντι της τοπικής εκτέλεσης στη συσκευή (on-device OCR). Για το σκοπό αυτό επιστρατεύτηκε το Google Cloud Vision API, το οποίο αξιοποιεί προεκπαιδευμένα, κορυφαία μοντέλα Μηχανικής Μάθησης (Machine Learning) της Google.

Η συγκεκριμένη υπηρεσία ενσωματώνει μια εξειδικευμένη μέθοδο ανάλυσης εγγράφων με την ονομασία DOCUMENT_TEXT_DETECTION. Η μέθοδος αυτή διαφοροποιείται από την απλή ανίχνευση κειμένου, καθώς είναι βελτιστοποιημένη για πυκνό κείμενο και έγγραφα. Αντιλαμβάνεται την ιεραρχική και γεωμετρική δομή του περιεχομένου, ομαδοποιώντας την πληροφορία σε διακριτά επίπεδα: περιοχές (blocks), παραγράφους (paragraphs), λέξεις (words) και χαρακτήρες (symbols) (Google Cloud, 2025).

Η δυνατότητα αυτή κρίνεται ως απολύτως απαραίτητη για την εφαρμογή προκειμένου να διατηρηθεί η χωρική συσχέτιση των δεδομένων, δηλαδή η σωστή αντιστοίχιση της αναγραφόμενης τιμής δίπλα στο αντίστοιχο προϊόν ή λεκτικό της απόδειξης. Αν και η προσέγγιση αυτή εισάγει την απαίτηση ενεργής σύνδεσης στο διαδίκτυο, η ανώτερη ακρίβεια που προσφέρει, ειδικά στην επεξεργασία της ελληνικής γλώσσας, την καθιστά τη βέλτιστη αρχιτεκτονική επιλογή για τη διασφάλιση της αξιοπιστίας των δεδομένων του χρήστη (Google Cloud, 2025).

3 Ανάλυση και Σχεδίαση Συστήματος

Πριν προχωρήσουμε στην υλοποίηση του κώδικα, είναι απαραίτητη η ανάλυση των απαιτήσεων και ο αρχιτεκτονικός σχεδιασμός της εφαρμογής. Στο κεφάλαιο αυτό προσδιορίζονται επακριβώς οι λειτουργικές και μη λειτουργικές προδιαγραφές του συστήματος, ενώ παράλληλα περιγράφονται οι αλληλεπιδράσεις του χρήστη με την εφαρμογή μέσω διαγραμμάτων περιπτώσεων χρήσης (Use Cases). Επιπρόσθετα, αποτυπώνεται ο σχεδιασμός της σχεσιακής βάσης δεδομένων και αναλύεται η εσωτερική ροή διεργασιών του υποσυστήματος Οπτικής Αναγνώρισης Χαρακτήρων (OCR) μέσω ενός διαγράμματος ακολουθίας (Sequence Diagram).

3.1 Απαιτήσεις Συστήματος

Οι προδιαγραφές του συστήματος κατηγοριοποιούνται σε δύο βασικούς άξονες: στις λειτουργικές απαιτήσεις, οι οποίες ορίζουν τη συμπεριφορά και τις δυνατότητες του λογισμικού, και στις μη λειτουργικές απαιτήσεις, οι οποίες καθορίζουν τα ποιοτικά και τεχνικά χαρακτηριστικά του.

3.1.1 Λειτουργικές Απαιτήσεις

Οι λειτουργικές απαιτήσεις (Functional Requirements) εξειδικεύουν τις υπηρεσίες και τις ενέργειες που το σύστημα πρέπει να είναι σε θέση να εκτελέσει κατά την αλληλεπίδρασή του με τον χρήστη. Για την εφαρμογή “Expense Tracker” ορίστηκαν οι παρακάτω:

1. **Προσθήκη Εξόδων (Χειροκίνητα & OCR):** Ο χρήστης έχει τη δυνατότητα να καταχωρήσει ένα νέο έξοδο χειροκίνητα (εισάγοντας το όνομα του καταστήματος, ποσό και κατηγορία) ή εναλλακτικά, να “σαρώνει” μια φυσική απόδειξη με την κάμερα. Στη δεύτερη περίπτωση, η εφαρμογή αναλαμβάνει να εξάγει αυτόματα τα απαραίτητα δεδομένα.
2. **Προβολή Ιστορικού και Στατιστική Ανάλυση:** Παρέχεται η δυνατότητα επισκόπησης του συνόλου των καταχωρισμένων εξόδων, οργανωμένων ιεραρχικά ανά μήνα και ανά έτος. Τα δεδομένα οπτικοποιούνται μέσω δυναμικών διαγραμμάτων, επιτρέποντας στον χρήστη να αναλύσει τις καταναλωτικές του συνήθειες.
3. **Διαχείριση Οικονομικών Ορίων (Budgeting):** Το σύστημα υποστηρίζει τον ορισμό ενός ανώτατου μηνιαίου προϋπολογισμού. Επιπλέον, προσφέρεται η δυνατότητα εξειδικευμένης κατανομής ορίων ξεχωριστά για κάθε κατηγορία δαπανών (π.χ. Σούπερ Μάρκετ, Μετακινήσεις).
4. **Παραμετροποίηση Περιβάλλοντος Εφαρμογής:** Ο χρήστης έχει τη δυνατότητα να προσαρμόσει τη διεπαφή βάσει των προτιμήσεών του. Αυτό περιλαμβάνει την επιλογή της γλώσσας, τον καθορισμό του προτιμώμενου νομίσματος συναλλαγών, καθώς και τη ρύθμιση του οπτικού θέματος (Φωτεινό, Σκοτεινό).
5. **Εξαγωγή Δεδομένων (Data Export):** Παρέχεται μηχανισμός εξαγωγής του τοπικού ιστορικού δαπανών σε δομημένη μορφή αρχείου CSV. Η λειτουργία αυτή διασφαλίζει τη δυνατότητα μεταφοράς των δεδομένων για περαιτέρω ανάλυση σε εξωτερικά εργαλεία (π.χ. υπολογιστικά φύλλα) ή για τη δημιουργία αντιγράφων ασφαλείας.
6. **Έλεγχος Κύκλου Ζωής Δεδομένων (Data Wipe):** Για λόγους ασφαλείας και διαχείρισης, το σύστημα ενσωματώνει τη λειτουργία ολικής και μη αναστρέψιμης διαγραφής του συνόλου των τοπικών δεδομένων και των ρυθμίσεων, επαναφέροντας την εφαρμογή στην αρχική της κατάσταση.

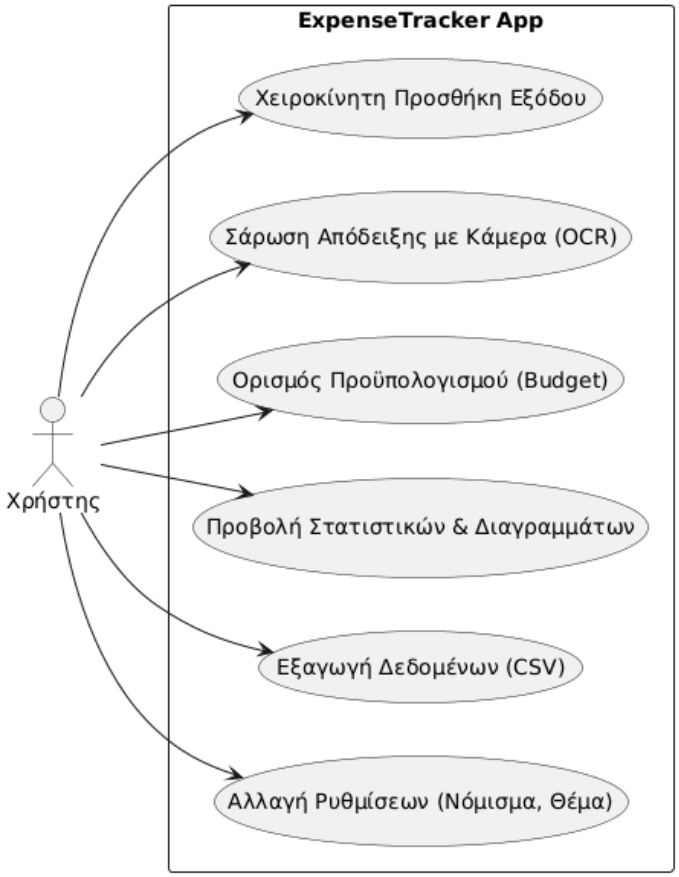
3.1.2 Μη Λειτουργικές Απαιτήσεις

Οι μη λειτουργικές απαιτήσεις (Non-Functional Requirements) καθορίζουν τα ποιοτικά κριτήρια, τους τεχνικούς περιορισμούς και τις προδιαγραφές απόδοσης του συστήματος:

1. **Βελτιστοποίηση Ροής Χρήσης (Frictionless Onboarding):** Η εφαρμογή έχει σχεδιαστεί με γνώμονα την ελαχιστοποίηση των εμποδίων εισόδου. Δεν απαιτείται η δημιουργία λογαριασμού (signup), η ταυτοποίηση (login) ή η επικοινωνία με απομακρυσμένους διακομιστές αυθεντικοποίησης. Ο χρήστης αποκτά άμεση πρόσβαση στο σύνολο των λειτουργιών αμέσως μετά την εγκατάσταση της εφαρμογής.
2. **Ιδιωτικότητα και Τοπικότητα Δεδομένων (Data Privacy & Locality):** Με γνώμονα την προστασία των ευαίσθητων οικονομικών δεδομένων, η αποθήκευση πραγματοποιείται αποκλειστικά στην τοπική σχεσιακή βάση δεδομένων της συσκευής. Κατά τη διαδικασία του OCR, η ψηφιακή εικόνα μεταφέρεται κρυπτογραφημένα στο Google Cloud Vision API με σκοπό την εφήμερη επεξεργασία και εξαγωγή του κειμένου, χωρίς να αποθηκεύεται μόνιμα ή να αρχειοθετείται στο υπολογιστικό νέφος.
3. **Ακρίβεια Υπολογιστικής Όρασης (Extraction Accuracy):** Η διαδικασία του OCR πρέπει να είναι όσο το δυνατόν πιο ακριβής στην αναγνώριση των ελληνικών χαρακτήρων και στον εντοπισμό του τελικού ποσού ανάμεσα στους υπόλοιπους αριθμούς της απόδειξης.
4. **Ευχρηστία και Συμμόρφωση Διεπαφής (Usability):** Το γραφικό περιβάλλον οφείλει να είναι λιτό, διαισθητικό και πλήρως εναρμονισμένο με τις επίσημες οδηγίες του Material Design 3. Επιπλέον, απαιτείται η απρόσκοπτη υποστήριξη και δυναμική εναλλαγή μεταξύ Φωτεινού (Light) και Σκοτεινού (Dark) θέματος.

3.2 Διαγράμματα Περιπτώσεων Χρήσης (Use Case Diagrams)

Τα διαγράμματα περιπτώσεων χρήσης (Use Case Diagrams) αναπαριστούν τον τρόπο με τον οποίο ο τελικός χρήστης αλληλεπιδρά με το σύστημα, αποτυπώνοντας τις βασικές λειτουργικότητες χωρίς να υπεισέρχονται σε τεχνικές λεπτομέρειες υλοποίησης. Στην περίπτωσή μας, υπάρχει ένας και μοναδικός "δράστης" (Actor), ο Χρήστης (User).



Διάγραμμα 3.1 Use Case Diagram

3.3 Σχεδιασμός Βάσης Δεδομένων (ER Diagram)

Δεδομένου ότι η εφαρμογή έχει σχεδιαστεί για αυστηρά προσωπική, τοπική χρήση χωρίς την απαίτηση συγχρονισμού πολλαπλών χρηστών ή τη διαχείριση πολύπλοκων σχεσιακών συσχετίσεων (όπως εξωτερικά κλειδιά σε πολλαπλούς πίνακες), ο σχεδιασμός του μοντέλου δεδομένων κρατήθηκε σκόπιμα αφαιρετικός και αποδοτικός.

Για την υλοποίηση της τοπικής αποθήκευσης μέσω της βιβλιοθήκης Room του Android, δημιουργήθηκε μία και μοναδική οντότητα, η ExpenseEntity, η οποία αντιστοιχεί στον πίνακα expenses_table της SQLite. Οι γενικές ρυθμίσεις του χρήστη (π.χ. γλώσσα, μηνιαίο budget) δεν επιβαρύνουν τη σχεσιακή βάση, αλλά χρησιμοποιείται η εγγενής, ελαφριά λύση αποθήκευσης προτιμήσεων (DataStore/Preferences) (Google Developers, 2025).

Η δομή και οι τύποι δεδομένων της οντότητας των εξόδων ορίζονται ως εξής:

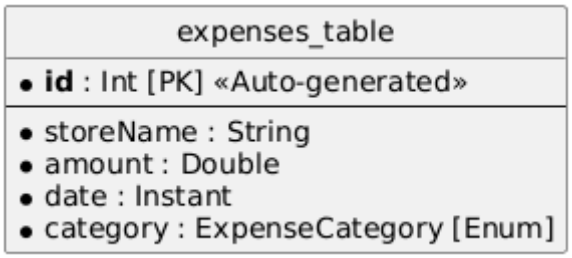
- **id (Integer):** Ακέραιος αριθμός που λειτουργεί ως το πρωτεύον κλειδί (PrimaryKey) του πίνακα και παράγεται αυτόματα (autoGenerate = true) από το σύστημα για κάθε νέα εγγραφή.
- **storeName (String):** Συμβολοσειρά που αποθηκεύει την επωνυμία της επιχείρησης ή του καταστήματος έκδοσης της απόδειξης.
- **amount (Double):** Δεκαδικός αριθμός κινητής υποδιαστολής διπλής ακρίβειας που αντιπροσωπεύει το

τελικό νομισματικό ποσό της δαπάνης.

- **date (Instant):** Σφραγίδα χρόνου που καταγράφει με ακρίβεια δευτερολέπτου την ημερομηνία και ώρα εκτέλεσης της συναλλαγής.
- **category (ExpenseCategory):** Τύπος απαρίθμησης (Enum) που κατηγοριοποιεί τη φύση της δαπάνης (π.χ. GROCERIES, TRAVEL, ENTERTAINMENT) και αποθηκεύεται στη βάση υπό μορφή συμβολοσειράς μέσω ειδικών μετατροπέων τύπου (Type Converters).

Πίνακας 3.1 Λεξικό δεδομένων πίνακα αποθήκευσης εξόδων (expenses_table)

Όνομα Πεδίου (FieldName)	Τύπος Δεδομένων (SQLite / Room)	Περιορισμοί (Constraints)	Περιγραφή / Συμπεριφορά
id	INTEGER / Int	PRIMARY KEY, AUTOINCREMENT	Μοναδικός αναγνωριστικός κωδικός για κάθε εγγραφή εξόδου.
storeName	TEXT / String	NOT NULL	Η επωνυμία της επιχείρησης ή του καταστήματος έκδοσης.
amount	REAL / Double	NOT NULL	Το τελικό νομισματικό ποσό της δαπάνης.
date	INTEGER / Instant	NOT NULL	Σφραγίδα χρόνου (Timestamp) της συναλλαγής (αποθηκεύεται ως Long μέσω TypeConverter).
category	TEXT / ExpenseCategory	NOT NULL	Τύπος απαρίθμησης (Enum) για τη φύση της δαπάνης (αποθηκεύεται ως String μέσω TypeConverter).



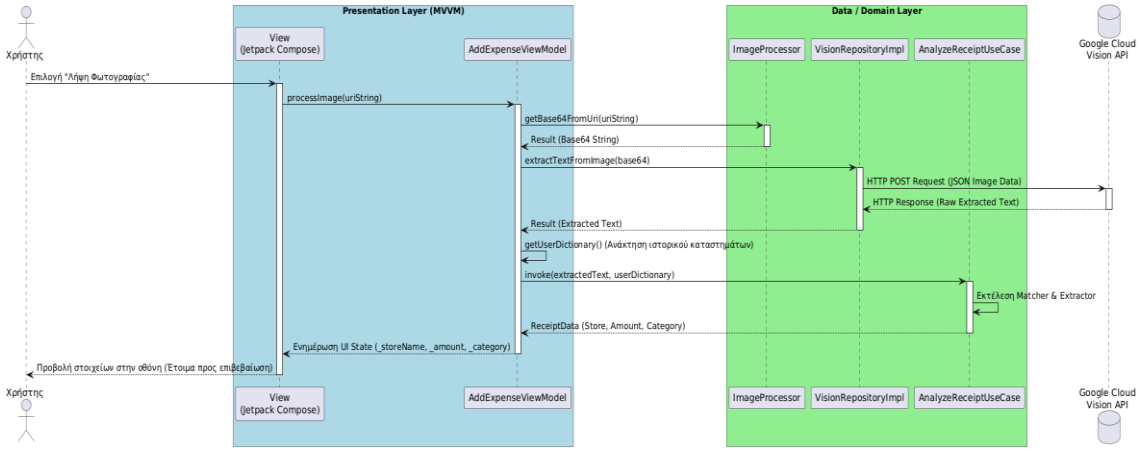
Διάγραμμα 3.2 ER Diagram

Η απουσία πολλαπλών σχεσιακών επιπέδων εξασφαλίζει τη βέλτιστη ταχύτητα εκτέλεσης ερωτημάτων (fast querying) χωρίς την υπολογιστική επιβάρυνση σύνθετων διεργασιών JOIN. Η αρχιτεκτονική αυτή επιλογή επιτρέπει στις αντιδραστικές ροές (Flows) της Room να ενημερώνουν ακαριαία τα γραφήματα της κύριας οθόνης κατά την εισαγωγή δεδομένων, διασφαλίζοντας μια απρόσκοπτη εμπειρία χρήστη.

3.4 Διάγραμμα Ακολουθίας (Sequence Diagram)

Για την πλήρη κατανόηση της δυναμικής συμπεριφοράς του συστήματος, κρίνεται απαραίτητη η αποτύπωση της ροής των μηνυμάτων στο χρόνο κατά την εκτέλεση της πλέον σύνθετης λειτουργίας της εφαρμογής: της αυτόματης εξαγωγής δεδομένων από αποδείξεις (OCR).

Το παρακάτω διάγραμμα ακολουθίας περιγράφει λεπτομερώς πώς αλληλεπιδρούν τα επιμέρους στοιχεία της αρχιτεκτονικής MVVM (View, ViewModel) με το Domain Layer (Use Cases) και το Data Layer (Repositories), ακολουθώντας πιστά την πραγματική ροή εκτέλεσης της συνάρτησης processImage() του συστήματος:



Διάγραμμα 3.3 OCR Sequence Diagram

4 Αρχιτεκτονική και Υλοποίηση

Στο κεφάλαιο αυτό παρουσιάζεται αναλυτικά η αρχιτεκτονική προσέγγιση που επιλέχθηκε για την ανάπτυξη της εφαρμογής “Expense Tracker”, καθώς και οι τεχνικές λεπτομέρειες της υλοποίησής της. Η εφαρμογή έχει δομηθεί με βάση τις αρχές της Καθαρής Αρχιτεκτονικής (Clean Architecture) σε συνδυασμό με το αρχιτεκτονικό πρότυπο MVVM (Model-View-ViewModel). Η προσέγγιση αυτή διασφαλίζει το διαχωρισμό των ευθυνών (Separation of Concerns) των επιμέρους υποσυστημάτων, καθιστώντας τον κώδικα εξαιρετικά δοκιμάσιμο (testable), επεκτάσιμο και εύκολα συντηρήσιμο.

4.1 Δομή του Έργου (Project Structure)

Η δομή των πακέτων (packages) της εφαρμογής ακολουθεί πιστά τα επίπεδα της Καθαρής Αρχιτεκτονικής. Ο κώδικας είναι οργανωμένος σε πέντε βασικά πακέτα κάτω από το κεντρικό namespace com.dionisispx.expensetracker:

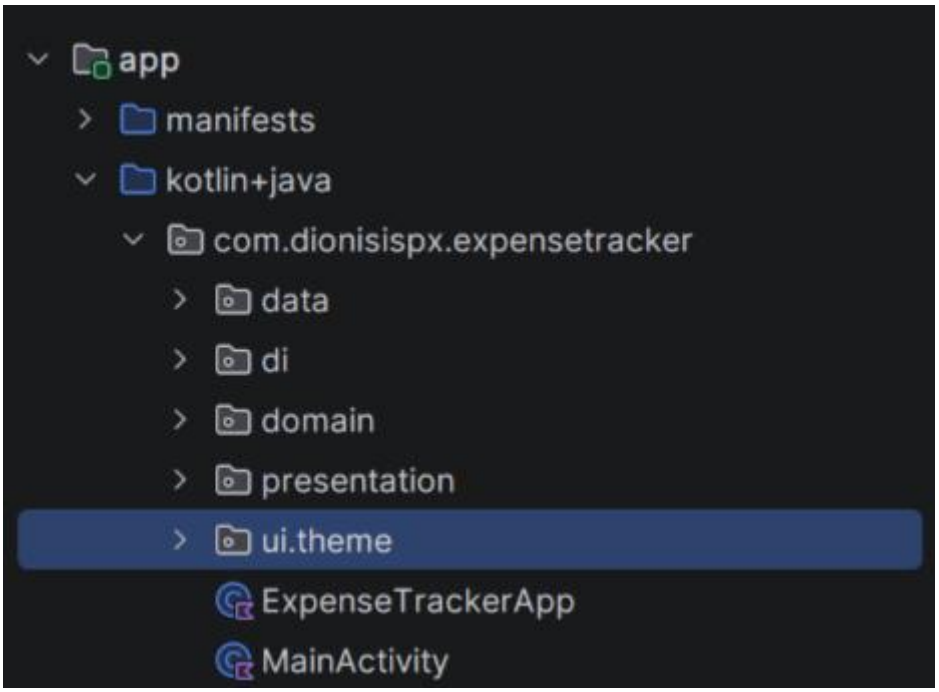
- **data:** Περιέχει την υλοποίηση της πρόσβασης στις πρωτογενείς πηγές δεδομένων της εφαρμογής. Περιλαμβάνει τις κλάσεις διασύνδεσης με την τοπική σχεσιακή βάση δεδομένων (Room Database/SQLite), την υλοποίηση των απομακρυσμένων δικτυακών κλήσεων (REST APIs) για το υποσύστημα OCR, καθώς και τη διαχείριση των τοπικών ρυθμίσεων μέσω του Preferences DataStore. Επιπλέον, στο πακέτο αυτό ενσωματώνονται οι μηχανισμοί μετατροπής δεδομένων (Mappers), οι οποίοι αναλαμβάνουν τη χαρτογράφηση μεταξύ των μοντέλων αποθήκευσης/δικτύου και των καθαρών επιχειρησιακών μοντέλων του Domain επιπέδου.

- **di (Dependency Injection):** Περιέχει τις μονάδες κώδικα (Modules) του framework Dagger Hilt. Οι μονάδες αυτές είναι υπεύθυνες για τη διαμόρφωση, την παραγωγή και την αυτόματη έγχυση των εξαρτήσεων σε ολόκληρη την εφαρμογή κατά το χρόνο μεταγλώττισης, ελαχιστοποιώντας τον χειροκίνητο συγχρονισμό των αντικειμένων.

- **domain:** Αποτελεί τον πυρήνα της επιχειρησιακής λογικής (Business Logic) της εφαρμογής και το πιο προστατευμένο επίπεδο της αρχιτεκτονικής. Περιλαμβάνει τα καθαρά μοντέλα δεδομένων, τους ορισμούς των διεπαφών των αποθετηρίων (Repository Interfaces), καθώς και τις περιπτώσεις χρήσης (Use Cases) του συστήματος, όπως ο αλγόριθμος ανάλυσης και επεξεργασίας δεδομένων OCR. Το πακέτο αυτό είναι πλήρως απομονωμένο και ανεξάρτητο από εξωτερικές βιβλιοθήκες, βάσεις δεδομένων ή frameworks διεπαφής.

- **presentation:** Αντιπροσωπεύει το επίπεδο παρουσίασης και είναι οργανωμένο με βάση τη δομή των λειτουργιών (feature-based packaging). Η προσέγγιση αυτή ομαδοποιεί τον κώδικα ανάλογα με την οθόνη ή τη λειτουργικότητα (π.χ. add_expense, budget, home, onboarding). Κάθε υποπακέτο περιέχει τα αντίστοιχα ViewModels για τη διαχείριση της κατάστασης (UI State), καθώς και τις συναρτήσεις Composables (οθόνες και επιμέρους UI components) που υλοποιούν το γραφικό περιβάλλον.

- **ui:** Περιέχει τις καθολικές ρυθμίσεις εμφάνισης, στυλ και συμπεριφοράς της εφαρμογής υπό το υποπακέτο ui.theme. Περιλαμβάνει τους κεντρικούς ορισμούς των χρωματικών παλετών, της τυπογραφίας και των σχημάτων, διαμορφώνοντας το ενιαίο θέμα της εφαρμογής σύμφωνα με τις προδιαγραφές του Material Design 3 Theme.



Εικόνα 4.1 Η δομή των πακέτων της εφαρμογής στο Android Studio

4.2 Το Επίπεδο Δεδομένων (Data Layer)

Το επίπεδο δεδομένων (Data Layer) είναι αποκλειστικά υπεύθυνο για την ανάκτηση, την τοπική αποθήκευση και το συντονισμό των δεδομένων από τις διάφορες πηγές. Αποτελείται από την τοπική βάση δεδομένων Room, την επικοινωνία με το Google Cloud Vision API μέσω της βιβλιοθήκης Retrofit (Square Inc., 2025), καθώς και το υποσύστημα αποθήκευσης προτιμήσεων και ρυθμίσεων μέσω του DataStore.

4.2.1 Τοπική Βάση Δεδομένων (Room Database)

Για την offline λειτουργία της εφαρμογής χρησιμοποιείται η βιβλιοθήκη Room, η οποία αποτελεί ένα επίπεδο αφαίρεσης (abstraction layer) πάνω από την SQLite.

Η οντότητα που αντιστοιχεί στον πίνακα της βάσης δεδομένων και ορίζει το σχήμα των αποθηκευμένων δεδομένων υλοποιείται μέσω της κλάσης κειμένου ExpenseEntity.

ExpenseEntity.kt

```kotlin
package com.dionisispx.expensetracker.data.local.entity

import ...

// Room database table for expenses
11 Usages
@Entity(tableName = "expenses_table")
data class ExpenseEntity(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val storeName: String,
    val amount: Double,
    val date: Instant,
    val category: ExpenseCategory
)

```

Εικόνα 4.2 Κώδικας ExpenseEntity.kt

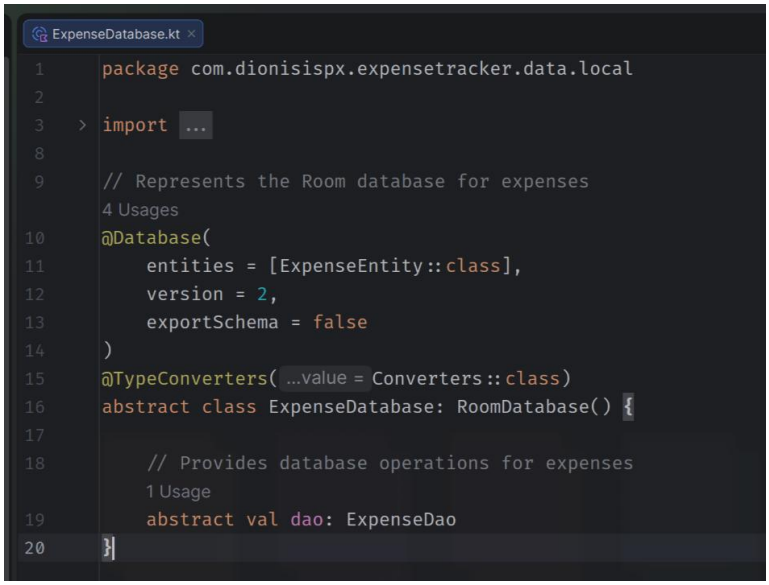
Οι λειτουργίες προσπέλασης, εισαγωγής και διαγραφής δεδομένων προδιαγράφονται στη διεπαφή ExpenseDao (Data Access Object). Στις μεθόδους ανάκτησης αξιοποιείται η δομή Flow των ασύγχρονων ροών της Kotlin, επιτρέποντας στο presentation layer να παρατηρεί (observe) τις μεταβολές των δεδομένων σε πραγματικό χρόνο, υλοποιώντας έτσι ένα αντιδραστικό (reactive) μοντέλο ενημέρωσης.

ExpenseDao.kt

```kotlin
package com.dionisispx.expensetracker.data.local

import ...

// Data access methods for expenses
5 Usages
@Dao
interface ExpenseDao {

    // Inserts or replaces an expense
    1 Usage
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertExpense(expense: ExpenseEntity): Long

    // Deletes an expense
    1 Usage
    @Delete
    suspend fun deleteExpense(expense: ExpenseEntity): Int

    // Retrieves all expenses
    1 Usage
    @Query( value = "SELECT * FROM expenses_table")
    fun getAllExpenses(): Flow<List<ExpenseEntity>>

    // Retrieves expenses within a date range
    1 Usage
    @Query( value = "SELECT * FROM expenses_table WHERE date BETWEEN :startDate AND :endDate ORDER BY date DESC")
    fun getExpensesByDateRange(startDate: Instant, endDate: Instant): Flow<List<ExpenseEntity>>

    // Deletes all expenses
    1 Usage
    @Query( value = "DELETE FROM expenses_table")
    suspend fun deleteAllExpenses()
}
```

Εικόνα 4.3 Κώδικας ExpenseDao.kt

Η σύνδεση των παραπάνω με το φυσικό αρχείο της βάσης δεδομένων γίνεται μέσω της αφηρημένης κλάσης ExpenseDatabase, η οποία κληρονομεί από την RoomDatabase και φέρει την επισημείωση (annotation) @Database. Στην κλάση αυτή δηλώνονται οι οντότητες, η έκδοση (version) της βάσης καθώς και οι TypeConverters που χρησιμοποιούνται.



Εικόνα 4.4 Κώδικας ExpenseDatabase.kt

Επειδή η SQLite δεν υποστηρίζει απευθείας τύπους δεδομένων όπως το Instant (ημερομηνία) και τα Enum (κατηγορία εξόδου), αναπτύχθηκε η βοηθητική κλάση Converters, η οποία αναλαμβάνει την αμφίδρομη μετατροπή των δεδομένων αυτών σε πρωτογενείς τύπους συμβατούς με την SQLite. Συγκεκριμένα, μετατρέπει το Instant σε Long και το Enum σε String κατά την εγγραφή, και την αντίστροφη ανακατασκευή τους κατά την ανάκτηση.

Converters.kt

```kotlin
package com.dionisispx.expensetracker.data.local

import ...

// Room database type converters
1 Usage
class Converters {

    @TypeConverter
    fun fromTimestamp(value: Long?): Instant? {
        return value?.let { Instant.ofEpochMilli(it) }
    }

    @TypeConverter
    fun dateToTimestamp(date: Instant?): Long? {
        return date?.toEpochMilli()
    }

    @TypeConverter
    fun fromExpenseCategory(value: String?): ExpenseCategory? {
        return value?.let { ExpenseCategory.fromDisplayName(it) }
    }

    @TypeConverter
    fun expenseCategoryToString(category: ExpenseCategory?): String? {
        return category?.displayName
    }
}
```

Εικόνα 4.5 Κώδικας Converters.kt

4.2.2 Διεπαφές Δικτύου (Remote API & Retrofit)

Για την υλοποίηση της αυτοματοποιημένης εξαγωγής δεδομένων (OCR), η εφαρμογή πραγματοποιεί ασύγχρονες δικτυακές κλήσεις προς το Google Cloud Vision API. Η δικτυακή αυτή υποδομή βασίζεται στη βιβλιοθήκη Retrofit, η οποία αποτελεί το βιομηχανικό πρότυπο τύπου type-safe HTTP client για το Android framework.

Το API απαιτεί την εκτέλεση ενός αιτήματος HTTP με τη μέθοδο POST, το οποίο μεταφέρει την ψηφιακή εικόνα κωδικοποιημένη σε μορφή συμβολοσειράς Base64. Τα μοντέλα δεδομένων που υποστηρίζουν αυτήν την επικοινωνία ορίζονται στο αρχείο VisionApiModels.kt και δομούνται ιεραρχικά ως εξής:

- **VisionRequest & AnnotateImageRequest:** Δομές που αναπαριστούν το σώμα του αιτήματος (HTTP Request Body). Μέσα σε αυτές εμπεριέχεται το αντικείμενο VisionImage, το οποίο συγκρατεί το Base64 string της εικόνας, καθώς και μια λίστα αντικειμένων τύπου Feature. Στη λίστα αυτή ορίζεται ο επιθυμητός τύπος ανίχνευσης, με προεπιλογή τη σταθερά DOCUMENT_TEXT_DETECTION, η οποία είναι ειδικά βελτιστοποιημένη για την ανάλυση πυκνού κειμένου και δομημένων εγγράφων, όπως οι αποδείξεις λιανικής.
- **VisionResponse & TextAnnotation:** Δομές που αναπαριστούν την απάντηση (HTTP Response) που επιστρέφει το API της Google. Από το σύνθετο σύνολο της επιστρεφόμενης πληροφορίας, η εφαρμογή απομονώνει τη λίστα textAnnotations. Το πρώτο στοιχείο αυτής της λίστας εμπεριέχει το πλήρες, ενοποιημένο ακατέργαστο κείμενο της απόδειξης που αναγνωρίστηκε, το οποίο στη συνέχεια μεταβιβάζεται στα Use Cases του Domain Layer για την εξαγωγή των τελικών στοιχείων.

VisionApiModels.kt

```kotlin
package com.dionisispx.expensetracker.data.remote

3 Usages
data class VisionRequest(val requests: List<AnnotateImageRequest>)
3 Usages
data class AnnotateImageRequest(val image: VisionImage, val features: List<Feature>)
3 Usages
data class VisionImage(val content: String) // Base64 encoded
3 Usages
data class Feature(val type: String = "DOCUMENT_TEXT_DETECTION")

1 Usage
data class VisionResponse(val responses: List<AnnotateImageResponse>?)
1 Usage
data class AnnotateImageResponse(val textAnnotations: List<TextAnnotation>?)
1 Usage
data class TextAnnotation(val description: String)
```

Εικόνα 4.6 Κώδικας VisionApiModels.kt

Η διεπαφή του δικτυακού API ορίζεται μέσω της κλάσης VisionApiService ως εξής:

VisionApiService.kt

```kotlin
package com.dionisispx.expensetracker.data.remote

import ...

// API service for Google Cloud Vision
5 Usages
interface VisionApi {
    // Annotates an image using the Vision API
    1 Usage
    @POST( value = "v1/images:annotate")
    suspend fun annotateImage(
        @Query( value = "key") apiKey: String,
        @Body request: VisionRequest
    ): VisionResponse
}
```

Εικόνα 4.7 Κώδικας VisionApiService.kt

Η υιοθέτηση των παραπάνω μοντέλων επιτρέπει στη βιβλιοθήκη Retrofit (σε συνεργασία με τον μετατροπέα GSON Converter) να εκτελεί αυτόματα τη σειριοποίηση (serialization) του αντικειμένου Request σε μορφή JSON, καθώς και την αποσειριοποίηση (deserialization) της απόκρισης JSON σε καθαρά, διαχειρίσιμα αντικείμενα της γλώσσας Kotlin.

4.2.3 Υλοποίηση των Αποθετηρίων (Repository Implementation)

Στο Data Layer πραγματοποιείται η υλοποίηση των διεπαφών (Repository Interfaces) που έχουν οριστεί στο Domain Layer. Η προσέγγιση αυτή αποτελεί την εφαρμογή της Αρχής Αντιστροφής Εξάρτησης (Dependency Inversion Principle), διασφαλίζοντας ότι τα ανώτερα επίπεδα λογικής παραμένουν πλήρως αποδεσμευμένα από τις τεχνικές λεπτομέρειες των υποδομών.

- **ExpenseRepositoryImpl:** Λειτουργεί ως ο συνδετικός κρίκος μεταξύ της επιχειρησιακής λογικής και του υποσυστήματος ExpenseDao. Κατά την ανάκτηση και την αποθήκευση των δεδομένων, η κλάση χρησιμοποιεί συναρτήσεις επέκτασης χαρτογράφησης (Mappers) που ορίζονται στο αρχείο ExpenseMapper.kt. Οι συναρτήσεις toDomain() και toEntity() αναλαμβάνουν την αμφίδρομη μετατροπή μεταξύ των αντικειμένων τύπου ExpenseEntity και των καθαρών επιχειρησιακών αντικειμένων Expense του Domain Layer. Η πρακτική αυτή αποτρέπει τη διαρροή λεπτομερειών του σχήματος της βάσης δεδομένων στα ανώτερα επίπεδα της αρχιτεκτονικής.
- **VisionRepositoryImpl:** Αναλαμβάνει την ενορχήστρωση της ροής δεδομένων του OCR. Δέχεται τη συμβολοσειρά Base64 που παράγεται από τον επεξεργαστή εικόνας, δομεί ιεραρχικά το αντικείμενο αίτηματος VisionRequest και εκτελεί την ασύγχρονη δικτυακή κλήση μέσω της VisionApiService, επιστρέφοντας το αναγνωρισμένο κείμενο.
- **UserPreferencesRepositoryImpl:** Παρατηρεί τις τοπικές ρυθμίσεις και τα οικονομικά όρια του χρήστη και υλοποιεί τους μηχανισμούς μόνιμης αποθήκευσης τους. Για τον σκοπό αυτό αξιοποιεί τη βιβλιοθήκη DataStore Preferences, η οποία προσφέρει ασφαλή, ασύγχρονη και ατομική (transactional) ροή αποθήκευσης key-value δεδομένων.

4.2.4 Προετοιμασία και Επεξεργασία Εικόνας (AndroidImageProcessor)

Πριν από την αποστολή των δεδομένων εικόνας στο Google Cloud Vision API, είναι κρίσιμο να προηγηθεί κατάλληλη προεπεξεργασία. Η ψηφιακή αυτή επεξεργασία υλοποιείται στην κλάση AndroidImageProcessor (εντός του πακέτου data/util) και κρίνεται ως απολύτως απαραίτητη για τη διασφάλιση της ορθότητας του OCR βάσει των ακόλουθων τριών αξόνων:

- **Διόρθωση Προσανατολισμού (Image Rotation):** Οι κάμερες των σύγχρονων κινητών τηλεφώνων αποθηκεύουν πληροφορίες προσανατολισμού στα μεταδεδομένα μορφότυπου EXIF (Exchangeable Image File Format) της εικόνας. Στην περίπτωση που μια φωτογραφία ληφθεί υπό κατακόρυφη διάταξη αλλά αποθηκευτεί οριζόντια, η διαδικασία οπτικής αναγνώρισης ενδέχεται να αποτύχει λόγω λανθασμένης στοίχισης. Κάνοντας χρήση της κλάσης ExifInterface, ο επεξεργαστής ανακτά τα μεταδεδομένα αυτά και, εάν απαιτείται, περιστρέφει το αντικείμενο Bitmap κατάλληλα (κατά 90, 180 ή 270 μοίρες) μέσω ενός γεωμετρικού πίνακα μετασχηματισμού (Matrix).
- **Κλιμάκωση και Συμπίεση Μεγέθους (Downscaling & Compression):** Οι σύγχρονοι αισθητήρες καμερών παράγουν αρχεία υψηλής ανάλυσης με μέγεθος αρκετών Megabytes. Η απευθείας διαβίβαση τέτοιων αρχείων στο API προκαλεί άσκοπη και υψηλή κατανάλωση δικτυακών δεδομένων, καθώς και σημαντική καθυστέρηση απόκρισης. Για το λόγο αυτό, ο image processor περιορίζει τη μέγιστη διάσταση της εικόνας στα προτεινόμενα 1024 pixels (διατηρώντας αυστηρά την αναλογία διαστάσεων - aspect ratio) και προβαίνει σε συμπίεση μορφότυπου JPEG με συντελεστή ποιότητας 80%.
- **Κωδικοποίηση Base64:** Στο τελικό στάδιο, ο συμπιεσμένος πίνακας δυαδικών δεδομένων (binary byte array) μετατρέπεται σε κωδικοποίηση κειμένου Base64. Η μετατροπή εκτελείται με τη χρήση της σταθεράς Base64.NO_WRAP, η οποία παρακάμπτει την προσθήκη χαρακτήρων αλλαγής γραμμής, ικανοποιώντας τις προδιαγραφές εισόδου του Vision API.

Δεδομένου ότι η επεξεργασία ψηφιακών εικόνων περιλαμβάνει υπολογιστικά εντατικές διεργασίες και λειτουργίες εισόδου/εξόδου, το σύνολο των ενεργειών εκτελείται ασύγχρονα στο παρασκήνιο. Αυτό επιτυγχάνεται μέσω της χρήσης του κατάλληλα βελτιστοποιημένου διαχειριστή νημάτων Dispatchers.IO της βιβλιοθήκης Kotlin Coroutines, διασφαλίζοντας ότι η κύρια ροή της διεπαφής (UI Thread) παραμένει ανεμπόδιστη.

4.3 Το Επίπεδο Επιχειρηματικής Λογικής (Domain Layer)

Το επίπεδο επιχειρησιακής λογικής (Domain Layer) αποτελεί τον πυρήνα της εφαρμογής. Σύμφωνα με τις αρχές της Καθαρής Αρχιτεκτονικής, συνιστά το πιο εσωτερικό επίπεδο αφαίρεσης, γεγονός που συνεπάγεται ότι διατηρεί πλήρη άγνοια σχετικά με εξωτερικές υποδομές, όπως οι βάσεις δεδομένων, τα πλαίσια διεπαφής (UI frameworks) ή οι βιβλιοθήκες δικτύου. Το Domain Layer περιλαμβάνει αποκλειστικά τα καθαρά επιχειρησιακά μοντέλα δεδομένων, τις αφηρημένες διεπαφές των αποθετηρίων και τις περιπτώσεις χρήσης (Use Cases) του συστήματος.

4.3.1 Μοντέλα Δεδομένων, Διεπαφές Αποθετηρίων και Βοηθητικά Interfaces

Στο επίπεδο αυτό προδιαγράφονται τα καθαρά επιχειρησιακά μοντέλα της εφαρμογής:

- **Expense:** Αναπαριστά μια οικονομική δαπάνη. Αποτελεί μια ανεξάρτητη κλάση δεδομένων (data class) της γλώσσας Kotlin, η οποία σε αντίθεση με την ExpenseEntity του Data Layer δεν φέρει μετα-δεδομένα ή annotations της βιβλιοθήκης Room.
- **ExpenseCategory:** Τύπος απαρίθμησης (Enum class) που ορίζει τις διαθέσιμες κατηγορίες εξόδων (π.χ. GROCERIES, FOOD_DRINK, SHOPPING) μαζί με τις αντίστοιχες εμφανιζόμενες ονομασίες τους.
- **ReceiptData:** Βοηθητική δομή δεδομένων που χρησιμοποιείται ως ενδιάμεσος φορέας μεταφοράς των στοιχείων που εξάγονται αυτοματοποιημένα από μια απόδειξη (επωνυμία επιχείρησης, τελικό νομισματικό ποσό, προτεινόμενη κατηγορία).

Expense.kt

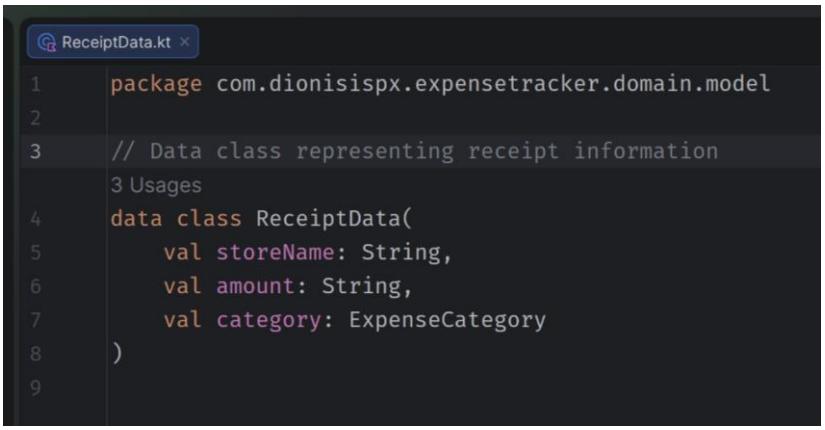
```kotlin
package com.dionisispx.expensetracker.domain.model

import java.time.Instant

// Represents an individual expense record
data class Expense(
    val id: Int = 0,
    val storeName: String,
    val amount: Double,
    val date: Instant,
    val category: ExpenseCategory
)
```

Εικόνα 4.8 Κώδικας Expense.kt

ExpenseCategory.kt

```kotlin
// Represents a category for an expense
enum class ExpenseCategory(val displayName: String) {
    8 Usages
    GROCERIES( displayName = "Groceries"),
    21 Usages
    FOOD_DRINK( displayName = "Food & Drink"),
    11 Usages
    TRANSPORT_FUEL( displayName = "Transport & Fuel"),
    14 Usages
    SHOPPING( displayName = "Shopping"),
    4 Usages
    ENTERTAINMENT( displayName = "Entertainment"),
    9 Usages
    BILLS_UTILITIES( displayName = "Bills & Utilities"),
    5 Usages
    HEALTH_FITNESS( displayName = "Health & Fitness"),
    2 Usages
    TRAVEL( displayName = "Travel"),
    2 Usages
    HOME( displayName = "Home"),
    2 Usages
    EDUCATION( displayName = "Education"),
    2 Usages
    PERSONAL_CARE( displayName = "Personal Care"),
    6 Usages
    OTHER( displayName = "Other");

    companion object {
        // Finds an expense category by its display name
        2 Usages
        fun fromDisplayName(name: String): ExpenseCategory {
            return entries.find { it.displayName == name } ?: OTHER
        }
    }
}
```

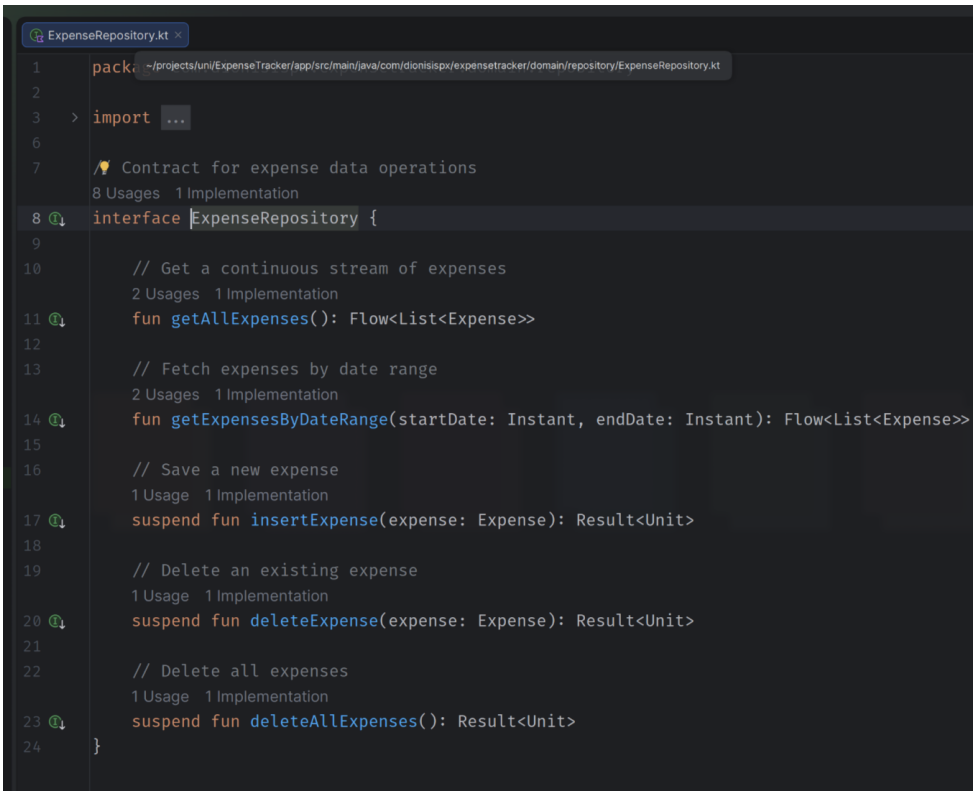
Εικόνα 4.9 Κώδικας ExpenseCategory.kt



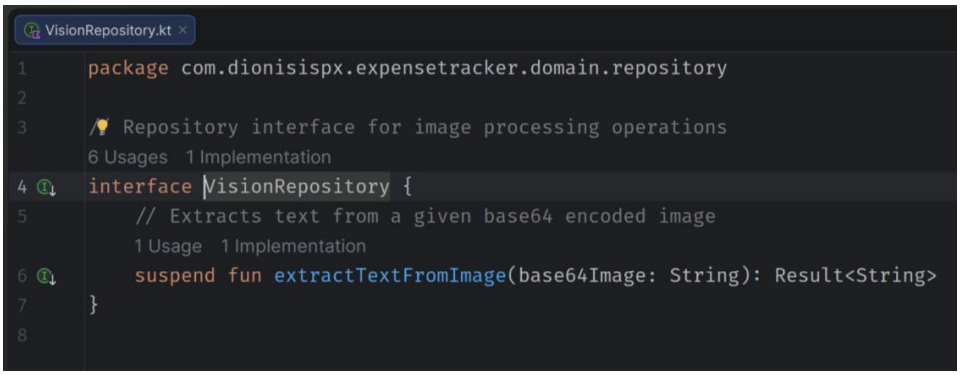
Εικόνα 4.10 Κώδικας ReceiptData.kt

Παράλληλα, στο Domain Layer ορίζονται οι αφηρημένες διεπαφές (Interfaces) των αποθετηρίων, οι οποίες καθορίζουν τα συμβόλαια λειτουργίας του συστήματος:

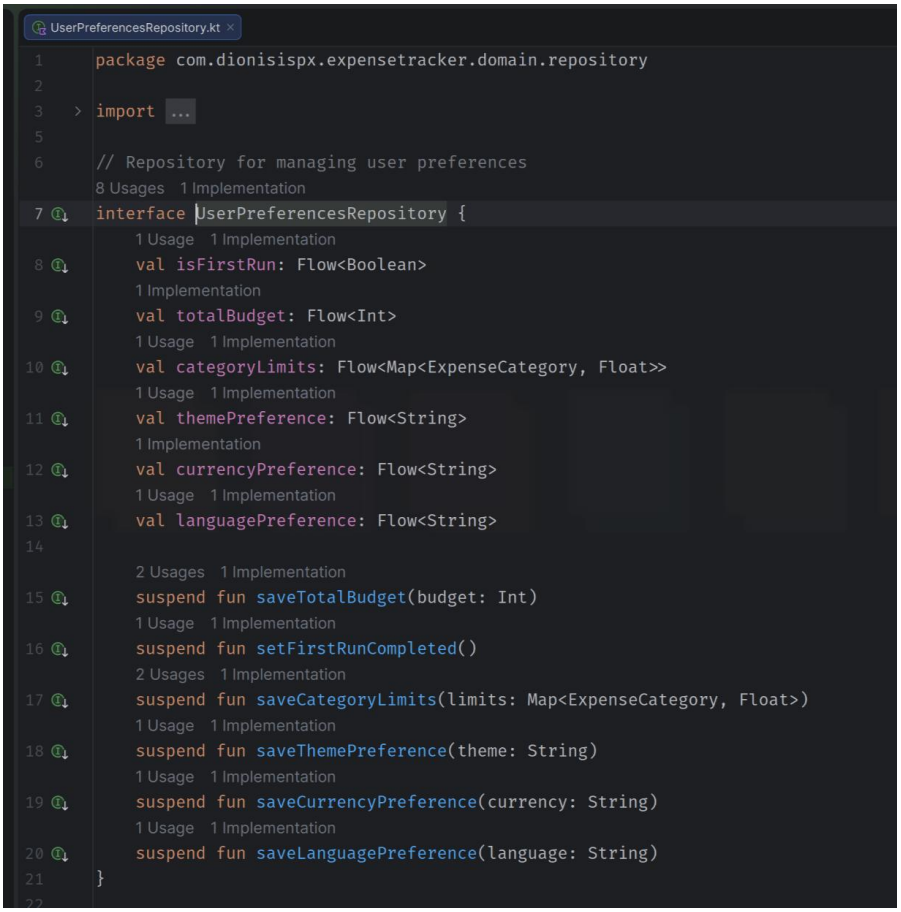
- **ExpenseRepository:** Καθορίζει τις λειτουργίες πρόσβασης στη βάση δεδομένων (εισαγωγή, διαγραφή, ανάκτηση εξόδων).
- **VisionRepository:** Προδιαγράφει τη μέθοδο ανάλυσης εικόνας για το OCR.
- **UserPreferencesRepository:** Καθορίζει τις λειτουργίες αποθήκευσης και ανάκτησης των τοπικών ρυθμίσεων του χρήστη.



Εικόνα 4.11 Κώδικας ExpenseRepository.kt

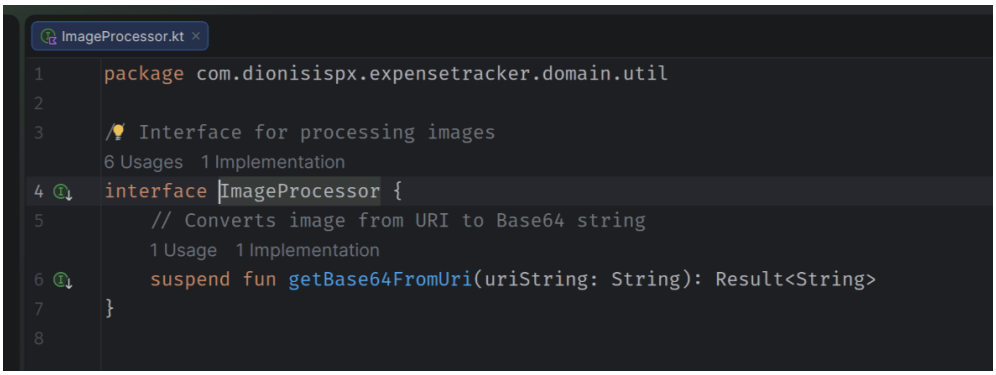


Εικόνα 4.12 Κώδικας VisionRepository.kt



Εικόνα 4.13 Κώδικας UserPreferencesRepository.kt

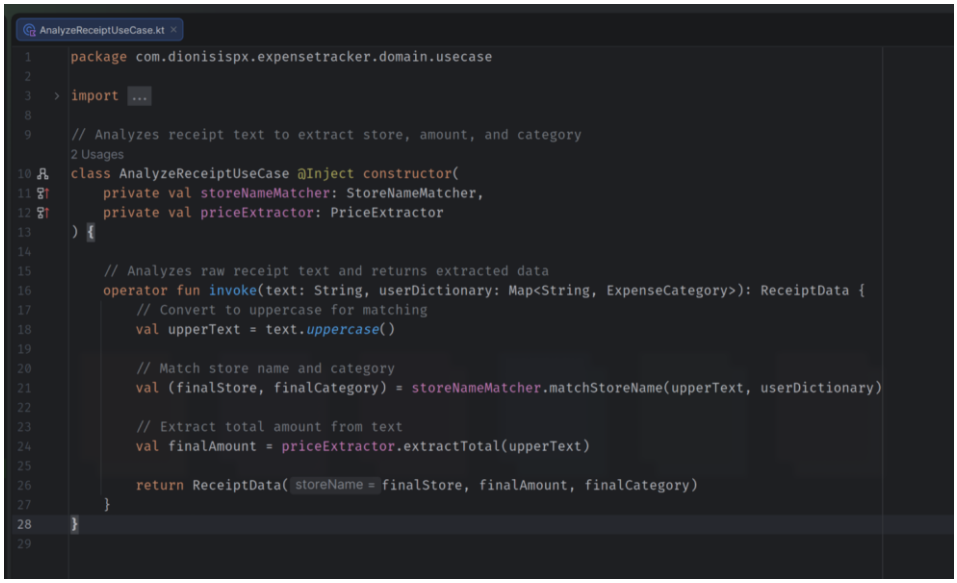
Τέλος, στο πλαίσιο του πακέτου domain/util, ορίζεται η βοηθητική διεπαφή ImageProcessor, η οποία περιλαμβάνει τη μέθοδο getBase64FromUri(uriString: String). Η σχεδίαση της στο Domain Layer διασφαλίζει ότι η επιχειρησιακή λογική γνωρίζει αποκλειστικά το λειτουργικό συμβόλαιο μετατροπής μιας εικόνας (από URI σε Base64 String), ενώ η πραγματική υλοποίηση με χρήση των χαμηλού επιπέδου βιβλιοθηκών του Android (Bitmap, ExifInterface και Context) ανατίθεται στο Data Layer (AndroidImageProcessor). Με τον τρόπο αυτό, εφαρμόζεται η Αντιστροφή Ελέγχου (Dependency Inversion), όπου το Domain Layer καθορίζει τι λειτουργίες απαιτούνται και το Data Layer εξειδικεύει το πώς αυτές θα εκτελεστούν.



Εικόνα 4.14 Κώδικας ImageProcessor.kt

4.3.2 Ο Αλγόριθμος OCR και η Επεξεργασία Κειμένου Απόδειξης (Receipt OCR Pipeline)

Το πιο απαιτητικό κομμάτι της εφαρμογής είναι η αυτόματη εξαγωγή πληροφοριών από το ακατέργαστο κείμενο της απόδειξης που επιστρέφει το Google Cloud Vision API. Η διαδικασία αυτή συντονίζεται από το AnalyzeReceiptUseCase και βασίζεται σε έναν εξελιγμένο αλγοριθμικό αγωγό (Pipeline) τριών διακριτών σταδίων: την κανονικοποίηση κειμένου (GreekTextNormalizer), την εύρεση καταστήματος (StoreNameMatcher) και την εξαγωγή του ποσού πληρωμής (PriceExtractor).



Εικόνα 4.15 Κώδικας AnalyzeReceiptUseCase.kt

4.3.2.1 Κανονικοποίηση Κειμένου και Υπολογισμός Ομοιότητας (GreekTextNormalizer)

Κατά την ψηφιοποίηση φυσικών αποδείξεων λιανικής, εμφανίζονται συστηματικά δύο κατηγορίες αλγοριθμικών προκλήσεων:

1. **Διακυμάνσεις Τονισμού και Πεζών/Κεφαλαίων:** Το API ενδέχεται να επιστρέψει χαρακτήρες με διαφορετικούς συντελεστές τονισμού ή ανάμεικτη χρήση πεζών και κεφαλαίων στοιχείων.
2. **Ομόγλυφοι Χαρακτήρες (Homoglyphs):** Αποτελεί το συχνότερο σφάλμα στην υπολογιστική όραση εγγράφων. Λόγω της οπτικής ταυτότητας πολλών ελληνικών και λατινικών χαρακτήρων (π.χ. το ελληνικό 'Α', 'Ε', 'Η' με τα λατινικά 'A', 'E', 'H'), οι μηχανές OCR τείνουν να συγχέουν τα character sets. Ως εκ τούτου, η λέξη «ΣΥΝΟΛΟ» αναγνωρίζεται συχνά ως «SYNOLO» (με χρήση λατινικών χαρακτήρων S, Y, O, L).

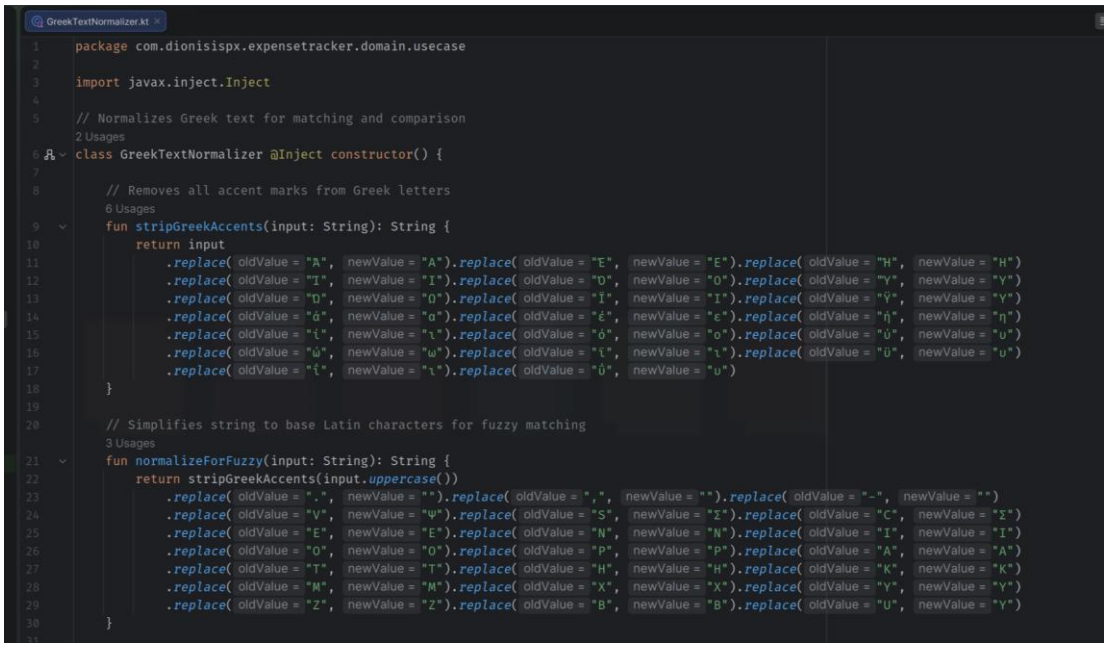
Η κλάση GreekTextNormalizer αντιμετωπίζει τις παραπάνω προκλήσεις μέσω δύο μεθοδολογικών συναρτήσεων:

- **stripGreekAccents:** Αφαιρεί με συστηματικό τρόπο όλους τους τόνους και τα διαλυτικά από το σύνολο των ελληνικών χαρακτήρων.
- **normalizeForFuzzy:** Μετατρέπει το γράμματα του κειμένου σε κεφαλαία, απαλείφει τα σημεία στίξης και αντικαθιστά τους λατινικούς χαρακτήρες με τους αντίστοιχους ελληνικούς ομόγλυφους (π.χ. το λατινικό 'B' μετατρέπεται στο ελληνικό 'Β' και το 'P' στο 'Ρ'), επιτυγχάνοντας πλήρη ομογενοποίηση των δεδομένων.

Για τη σύγκριση συμβολοσειρών υπό την παρουσία τυπογραφικών ή αναγνωστικών σφαλμάτων, επιστρατεύεται ο αλγόριθμος Levenshtein Distance (Edit Distance). Η μέθοδος similarity με την βοήθεια της levenshteinDistance υπολογίζει ένα ποσοστό ομοιότητας μεταξύ δύο λέξεων s1, s2 βάσει του ελάχιστου αριθμού ενεργειών (εισαγωγή, διαγραφή, αντικατάσταση χαρακτήρα) που απαιτούνται για να μετατραπεί η μία στην άλλη και ορίζεται μαθηματικά ως εξής (Levenshtein, 1966):

$$Similarity(s_1, s_2) = (\max(|s_1|, |s_2|) - Levenshtein(s_1, s_2))/\max(|s_1|, |s_2|)$$

Εικόνα 4.16 Μαθηματική διατύπωση Similarity



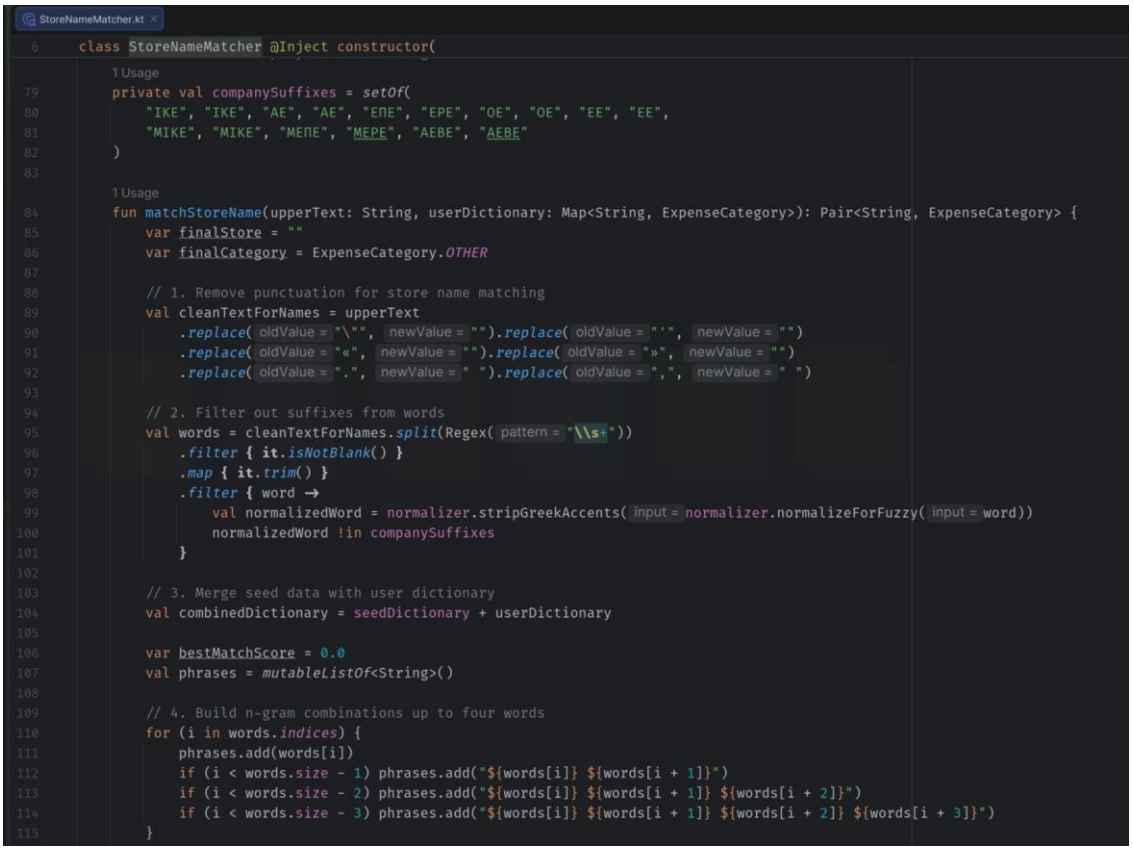
```kotlin
package com.dionisispx.expensetracker.domain.usecase

import javax.inject.Inject

// Normalizes Greek text for matching and comparison
2 Usages
class GreekTextNormalizer @Inject constructor() {

    // Removes all accent marks from Greek letters
    6 Usages
    fun stripGreekAccents(input: String): String {
        return input
            .replace( oldValue = "Α", newValue = "A").replace( oldValue = "Ε", newValue = "E").replace( oldValue = "Η", newValue = "H")
            .replace( oldValue = "Ι", newValue = "I").replace( oldValue = "Ο", newValue = "O").replace( oldValue = "Υ", newValue = "Y")
            .replace( oldValue = "Ό", newValue = "Ο").replace( oldValue = "Ί", newValue = "Ι").replace( oldValue = "Ύ", newValue = "Υ")
            .replace( oldValue = "ά", newValue = "α").replace( oldValue = "έ", newValue = "ε").replace( oldValue = "ή", newValue = "η")
            .replace( oldValue = "ί", newValue = "ι").replace( oldValue = "ό", newValue = "ο").replace( oldValue = "ύ", newValue = "υ")
            .replace( oldValue = "ώ", newValue = "ω").replace( oldValue = "ΐ", newValue = "ι").replace( oldValue = "ΰ", newValue = "υ")
            .replace( oldValue = "ϊ", newValue = "ι").replace( oldValue = "ϋ", newValue = "υ")
    }

    // Simplifies string to base Latin characters for fuzzy matching
    3 Usages
    fun normalizeForFuzzy(input: String): String {
        return stripGreekAccents(input.uppercase())
            .replace( oldValue = ".", newValue = "").replace( oldValue = ",", newValue = "").replace( oldValue = "-", newValue = "")
            .replace( oldValue = "V", newValue = "Υ").replace( oldValue = "S", newValue = "Σ").replace( oldValue = "C", newValue = "Σ")
            .replace( oldValue = "E", newValue = "Ε").replace( oldValue = "N", newValue = "Ν").replace( oldValue = "I", newValue = "Ι")
            .replace( oldValue = "O", newValue = "Ο").replace( oldValue = "P", newValue = "Ρ").replace( oldValue = "A", newValue = "Α")
            .replace( oldValue = "T", newValue = "Τ").replace( oldValue = "H", newValue = "Η").replace( oldValue = "K", newValue = "Κ")
            .replace( oldValue = "M", newValue = "Μ").replace( oldValue = "X", newValue = "Χ").replace( oldValue = "Y", newValue = "Υ")
            .replace( oldValue = "Z", newValue = "Ζ").replace( oldValue = "B", newValue = "Β").replace( oldValue = "U", newValue = "Υ")
    }
```

Εικόνα 4.17 Κώδικας similarity της GreekTextNormalizer

4.3.2.2 Εύρεση Καταστήματος και Κατηγορίας (StoreNameMatcher)

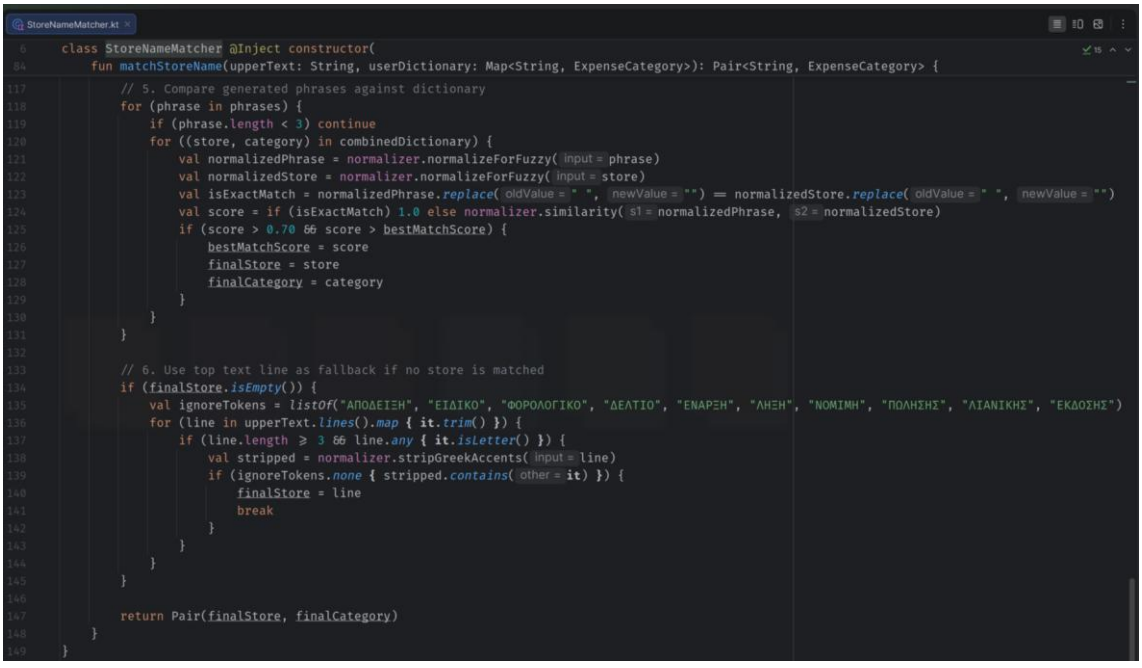
Η κλάση StoreNameMatcher αναλαμβάνει την ταυτοποίηση της επωνυμίας του εκδότη της απόδειξης και την απόδοση μιας προτεινόμενης κατηγορίας εξόδου, εκτελώντας την ακόλουθη διαδικασία:

1. **Φιλτράρισμα Εταιρικών Καταλήξεων (Noise Filtering):** Αφαιρούνται λέξεις θορύβου που αφορούν εταιρικές μορφές (όπως «Α.Ε.», «Ι.Κ.Ε.», «Ε.Π.Ε.», «Ο.Ε.»), καθώς η παρουσία τους δυσχεραίνει τη διαδικασία σύγκρισης.
2. **Παραγωγή N-Grams:** Το κείμενο τεμαχίζεται σε διακριτές λέξεις και παράγονται συνδυασμοί συνεχόμενων λέξεων (από 1-gram έως 4-gram). Η τεχνική αυτή επιτρέπει την αναγνώριση σύνθετων επωνυμιών που αποτελούνται από πολλαπλές λέξεις (π.χ. «COFFEE ISLAND» ή «ΑΒ ΒΑΣΙΛΟΠΟΥΛΟΣ»).
3. **Ασαφής Ταυτοποίηση (Fuzzy Matching):** Κάθε παραγόμενο N-gram συγκρίνεται μέσω της απόστασης Levenshtein με ένα στατικό λεξικό γνωστών εμπόρων (seedDictionary), καθώς και με το δυναμικό, εξατομικευμένο λεξικό που δημιουργεί ο χρήστης βάσει του ιστορικού του (userDictionary). Εάν ο δείκτης ομοιότητας υπερβαίνει το 70% (0.70), η επωνυμία θεωρείται ότι ταυτοποιήθηκε με επιτυχία.
4. **Εφεδρικός Μηχανισμός (Fallback):** Σε περίπτωση αδυναμίας εντοπισμού γνωστής επωνυμίας, ο αλγόριθμος επιλέγει ως όνομα καταστήματος την πρώτη γραμμή της απόδειξης που περιέχει αλφαβητικά στοιχεία και στερείται λέξεων θορύβου (όπως «ΑΠΟΔΕΙΞΗ» ή «ΦΟΡΟΛΟΓΙΚΟ»), εκμεταλλευόμενος το γεγονός ότι η επωνυμία τυπώνεται στην κορυφή της απόδειξης.



```kotlin
class StoreNameMatcher @Inject constructor(

    1 Usage
    private val companySuffixes = setOf(
        "ΙΚΕ", "ΙΚΕ", "ΑΕ", "ΑΕ", "ΕΠΕ", "ΕΡΕ", "ΟΕ", "ΟΕ", "ΕΕ", "ΕΕ",
        "ΜΙΚΕ", "ΜΙΚΕ", "ΜΕΠΕ", "ΜΕΡΕ", "ΑΕΒΕ", "ΑΕΒΕ"
    )

    1 Usage
    fun matchStoreName(upperText: String, userDictionary: Map<String, ExpenseCategory>): Pair<String, ExpenseCategory> {
        var finalStore = ""
        var finalCategory = ExpenseCategory.OTHER

        // 1. Remove punctuation for store name matching
        val cleanTextForNames = upperText
            .replace( oldValue = "\"", newValue = "").replace( oldValue = "'", newValue = "")
            .replace( oldValue = "«", newValue = "").replace( oldValue = "»", newValue = "")
            .replace( oldValue = ".", newValue = " ").replace( oldValue = ",", newValue = " ")

        // 2. Filter out suffixes from words
        val words = cleanTextForNames.split(Regex( pattern = "\\s+"))
            .filter { it.isNotBlank() }
            .map { it.trim() }
            .filter { word →
                val normalizedWord = normalizer.stripGreekAccents( input = normalizer.normalizeForFuzzy( input = word))
                normalizedWord !in companySuffixes
            }

        // 3. Merge seed data with user dictionary
        val combinedDictionary = seedDictionary + userDictionary

        var bestMatchScore = 0.0
        val phrases = mutableListOf<String>()

        // 4. Build n-gram combinations up to four words
        for (i in words.indices) {
            phrases.add(words[i])
            if (i < words.size - 1) phrases.add("${words[i]} ${words[i + 1]}")
            if (i < words.size - 2) phrases.add("${words[i]} ${words[i + 1]} ${words[i + 2]}")
            if (i < words.size - 3) phrases.add("${words[i]} ${words[i + 1]} ${words[i + 2]} ${words[i + 3]}")
        }
```

Εικόνα 4.18 Κώδικας matchStoreName της StoreNameMatcher (1)



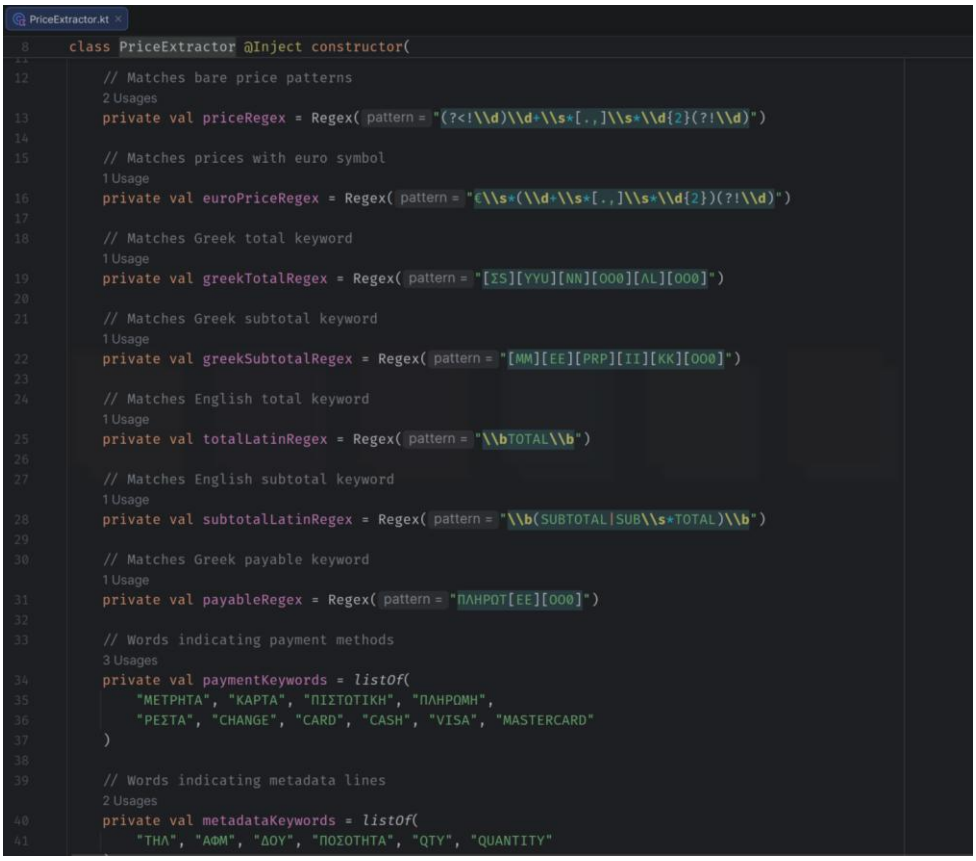
```kotlin
class StoreNameMatcher @Inject constructor(
    fun matchStoreName(upperText: String, userDictionary: Map<String, ExpenseCategory>): Pair<String, ExpenseCategory> {
        // 5. Compare generated phrases against dictionary
        for (phrase in phrases) {
            if (phrase.length < 3) continue
            for ((store, category) in combinedDictionary) {
                val normalizedPhrase = normalizer.normalizeForFuzzy( input = phrase)
                val normalizedStore = normalizer.normalizeForFuzzy( input = store)
                val isExactMatch = normalizedPhrase.replace( oldValue = " ", newValue = "") == normalizedStore.replace( oldValue = " ", newValue = "")
                val score = if (isExactMatch) 1.0 else normalizer.similarity( s1 = normalizedPhrase, s2 = normalizedStore)
                if (score > 0.70 && score > bestMatchScore) {
                    bestMatchScore = score
                    finalStore = store
                    finalCategory = category
                }
            }
        }

        // 6. Use top text line as fallback if no store is matched
        if (finalStore.isEmpty()) {
            val ignoreTokens = listOf("ΑΠΟΔΕΙΞΗ", "ΕΙΔΙΚΟ", "ΦΟΡΟΛΟΓΙΚΟ", "ΔΕΛΤΙΟ", "ΕΝΑΡΞΗ", "ΛΗΞΗ", "ΝΟΜΙΜΗ", "ΠΩΛΗΣΗΣ", "ΑΙΑΝΙΚΗΣ", "ΕΚΔΟΣΗΣ")
            for (line in upperText.lines().map { it.trim() }) {
                if (line.length ≥ 3 && line.any { it.isLetter() }) {
                    val stripped = normalizer.stripGreekAccents( input = line)
                    if (ignoreTokens.none { stripped.contains( other = it) }) {
                        finalStore = line
                        break
                    }
                }
            }
        }

        return Pair(finalStore, finalCategory)
    }
}
```

Εικόνα 4.19 Κώδικας matchStoreName της StoreNameMatcher (2)

4.3.2.3 Εξαγωγή Ποσού (PriceExtractor)

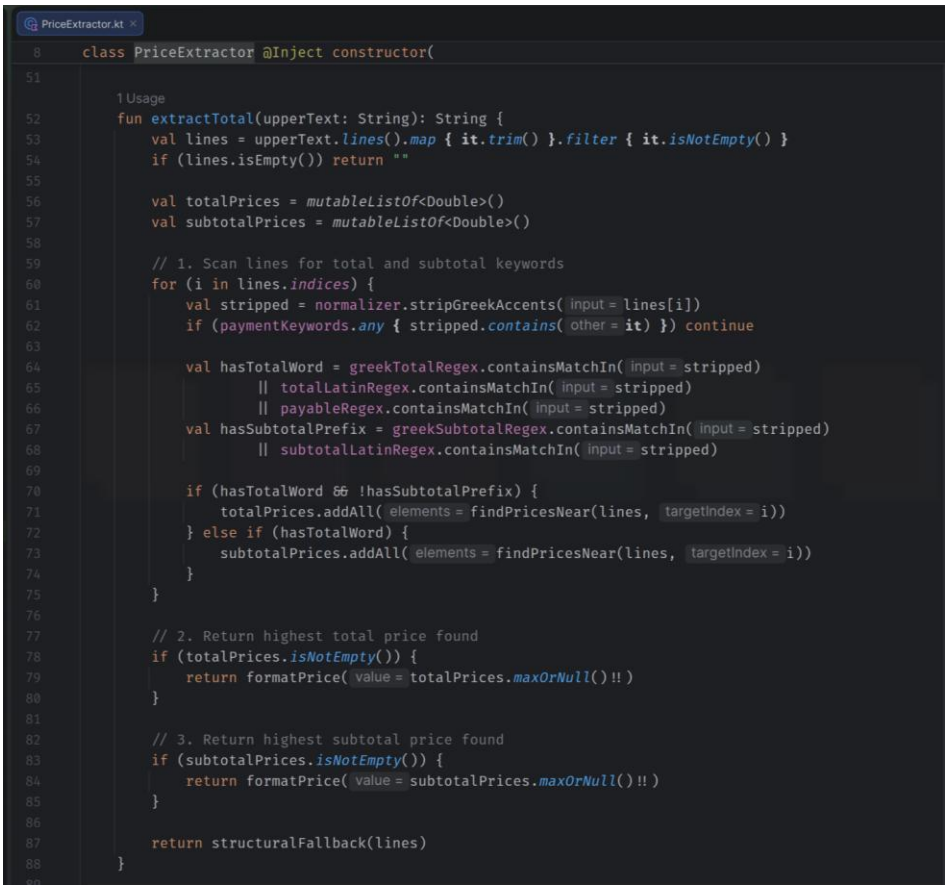
Η κλάση PriceExtractor αναλαμβάνει τον εντοπισμό και την απομόνωση του τελικού πληρωτέου ποσού. Η διεργασία αυτή παρουσιάζει υψηλό βαθμό δυσκολίας, καθώς οι αποδείξεις περιέχουν πλήθος παρεμφερών αριθμητικών τιμών (π.χ. ΑΦΜ, ποσότητες, τιμές μονάδας, αξίες ΦΠΑ, αθροίσματα). Για την επίλυση του προβλήματος, ο αλγόριθμος εκτελεί τα εξής βήματα:

1. **Σάρωση Λέξεων-Κλειδιών (Keyword Scanning):** Το σύστημα εξετάζει τις γραμμές του κειμένου για τον εντοπισμό λεκτικών σημαδιών που υποδηλώνουν το τελικό σύνολο (π.χ. «ΣΥΝΟΛΟ», «TOTAL», «ΠΛΗΡΩΤΕΟ»). Οι χρησιμοποιούμενες κανονικές εκφράσεις (Regular Expressions) έχουν σχεδιαστεί ώστε να είναι ανθεκτικές σε θόρυβο (π.χ. η έκφραση [ΣS][ΥYU][ΝN][ΟO0][ΛL][ΟO0] ταυτοποιεί επιτυχώς τη λέξη ακόμη και αν αυτή φέρει λατινικά στοιχεία ή το ψηφίο μηδέν αντί για το γράμμα Όμικρον).



Εικόνα 4.20 Κανονικές εκφράσεις και λέξεις-κλειδιά για τον εντοπισμό τιμών και συνόλων της PriceExtractor

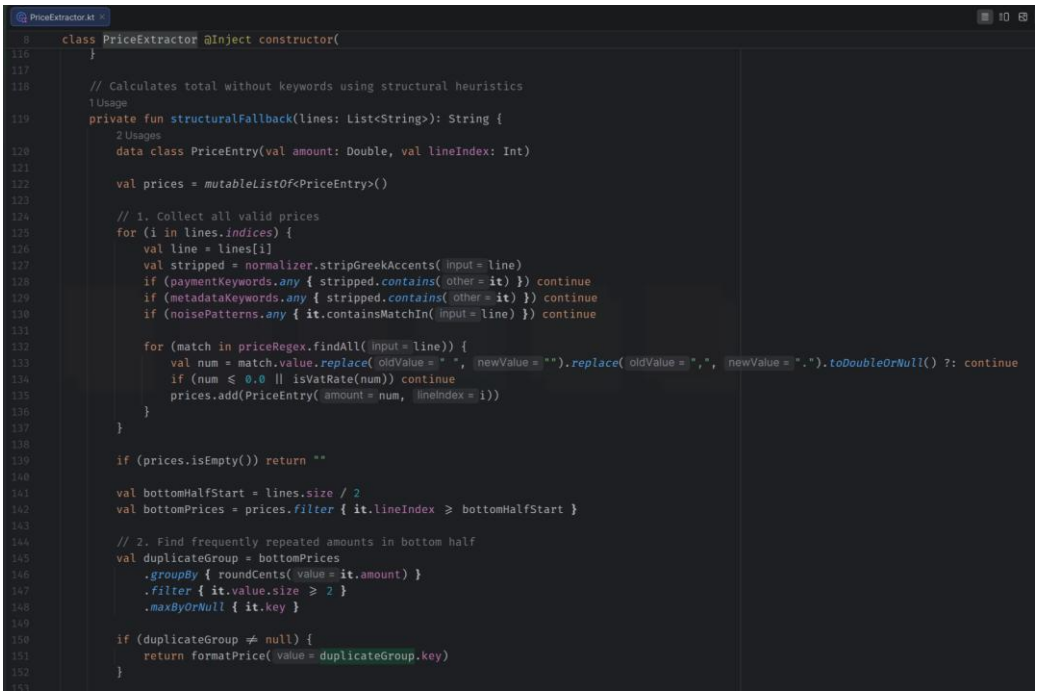
2. **Ανάκτηση Γειτονικών Τιμών (findPricesNear):** Μόλις εντοπιστεί μια λέξη-κλειδί, ο αλγόριθμος αναζητά δεκαδικούς αριθμούς εντός της ίδιας γραμμής, καθώς και των επόμενων δύο γραμμών, καλύπτοντας περιπτώσεις όπου το ποσό τυπώνεται ακριβώς κάτω από τη σήμανση του συνόλου.



Εικόνα 4.21 Η κεντρική μέθοδος extractTotal της PriceExtractor για την εύρεση του πληρωτέου ποσού

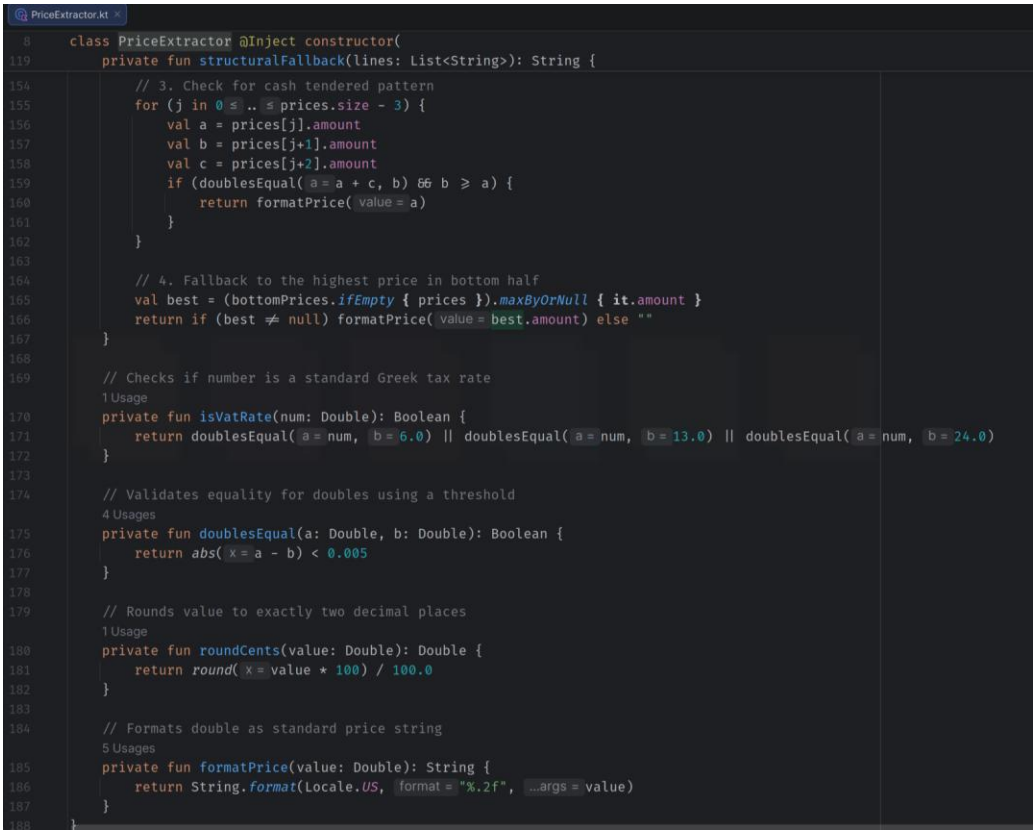
3. **Δομικός Εφεδρικός Αλγόριθμος (Structural Fallback):** Στην περίπτωση που δεν ανιχνευθούν λέξεις-κλειδιά (λόγω αποκοπής ή κακής εκτύπωσης), ενεργοποιείται μια σειρά από προηγμένους ευρετικούς κανόνες (Heuristics):

   i. **Φιλτράρισμα ΦΠΑ:** Απαλείφονται αυτόματα αριθμητικές τιμές που συμπίπτουν με τους θεσμοθετημένους συντελεστές ΦΠΑ στην Ελλάδα (6.0, 13.0, 24.0) προς αποφυγή ψευδώς θετικών αποτελεσμάτων.
   ii. **Ευρετική Επαναλαμβανόμενων Τιμών:** Εάν μια συγκεκριμένη δεκαδική τιμή εντοπιστεί περισσότερες από δύο φορές στο κάτω μισό της απόδειξης, συγκεντρώνει υψηλή πιθανότητα να αποτελεί το τελικό σύνολο.
   iii. **Μαθηματική Ευρετική (Cash & Change):** Αναζητείται μια μαθηματική τριάδα τιμών a, b, c που να ικανοποιεί τη σχέση a + c = b (Σύνολο + Ρέστα = Μετρητά που δόθηκαν). Εάν η σχέση επαληθευτεί, η τιμή a επιστρέφεται ως το πληρωτέο ποσό.
   iv. **Κριτήριο Μέγιστης Τιμής:** Ως έσχατη λύση (fallback), ο αλγόριθμος απομονώνει και επιλέγει τη μεγαλύτερη αριθμητική τιμή που εμφανίζεται στο κάτω τμήμα της απόδειξης.



```kotlin
class PriceExtractor @Inject constructor(
    }

    // Calculates total without keywords using structural heuristics
    1 Usage
    private fun structuralFallback(lines: List<String>): String {
        2 Usages
        data class PriceEntry(val amount: Double, val lineIndex: Int)

        val prices = mutableListOf<PriceEntry>()

        // 1. Collect all valid prices
        for (i in lines.indices) {
            val line = lines[i]
            val stripped = normalizer.stripGreekAccents( input = line)
            if (paymentKeywords.any { stripped.contains( other = it) }) continue
            if (metadataKeywords.any { stripped.contains( other = it) }) continue
            if (noisePatterns.any { it.containsMatchIn( input = line) }) continue

            for (match in priceRegex.findAll( input = line)) {
                val num = match.value.replace( oldValue = " ", newValue = "").replace( oldValue = ",", newValue = ".").toDoubleOrNull() ?: continue
                if (num ≤ 0.0 || isVatRate(num)) continue
                prices.add(PriceEntry( amount = num, lineIndex = i))
            }
        }

        if (prices.isEmpty()) return ""

        val bottomHalfStart = lines.size / 2
        val bottomPrices = prices.filter { it.lineIndex ≥ bottomHalfStart }

        // 2. Find frequently repeated amounts in bottom half
        val duplicateGroup = bottomPrices
            .groupBy { roundCents( value = it.amount) }
            .filter { it.value.size ≥ 2 }
            .maxByOrNull { it.key }

        if (duplicateGroup ≠ null) {
            return formatPrice( value = duplicateGroup.key)
        }
```

Εικόνα 4.22 Η μέθοδος structuralFallback της PriceExtractor για τον εντοπισμό ποσού βάσει ευρετικών κανόνων (1)



```kotlin
class PriceExtractor @Inject constructor(
    private fun structuralFallback(lines: List<String>): String {
        // 3. Check for cash tendered pattern
        for (j in 0 ≤ .. ≤ prices.size - 3) {
            val a = prices[j].amount
            val b = prices[j+1].amount
            val c = prices[j+2].amount
            if (doublesEqual( a = a + c, b) && b ≥ a) {
                return formatPrice( value = a)
            }
        }

        // 4. Fallback to the highest price in bottom half
        val best = (bottomPrices.ifEmpty { prices }).maxByOrNull { it.amount }
        return if (best ≠ null) formatPrice( value = best.amount) else ""
    }

    // Checks if number is a standard Greek tax rate
    1 Usage
    private fun isVatRate(num: Double): Boolean {
        return doublesEqual( a = num, b = 6.0) || doublesEqual( a = num, b = 13.0) || doublesEqual( a = num, b = 24.0)
    }

    // Validates equality for doubles using a threshold
    4 Usages
    private fun doublesEqual(a: Double, b: Double): Boolean {
        return abs( x = a - b) < 0.005
    }

    // Rounds value to exactly two decimal places
    1 Usage
    private fun roundCents(value: Double): Double {
        return round( x = value * 100) / 100.0
    }

    // Formats double as standard price string
    5 Usages
    private fun formatPrice(value: Double): String {
        return String.format(Locale.US, format = "%.2f", ...args = value)
    }
}
```

Εικόνα 4.23 Η μέθοδος structuralFallback της PriceExtractor για τον εντοπισμό ποσού βάσει ευρετικών κανόνων (2)

4.4 Το Επίπεδο Παρουσίασης (Presentation & UI Layer)

Το επίπεδο παρουσίασης (Presentation Layer) είναι υπεύθυνο για την υλοποίηση της διαδραστικής διεπαφής με τον τελικό χρήστη και τη διαχείριση των οπτικών συμβάντων. Η αρχιτεκτονική του βασίζεται στο πρότυπο MVVM (Model-View-ViewModel) σε συνδυασμό με το Jetpack Compose της Google, υιοθετώντας πλήρως το δηλωτικό παράδειγμα σχεδιασμού (Declarative UI).

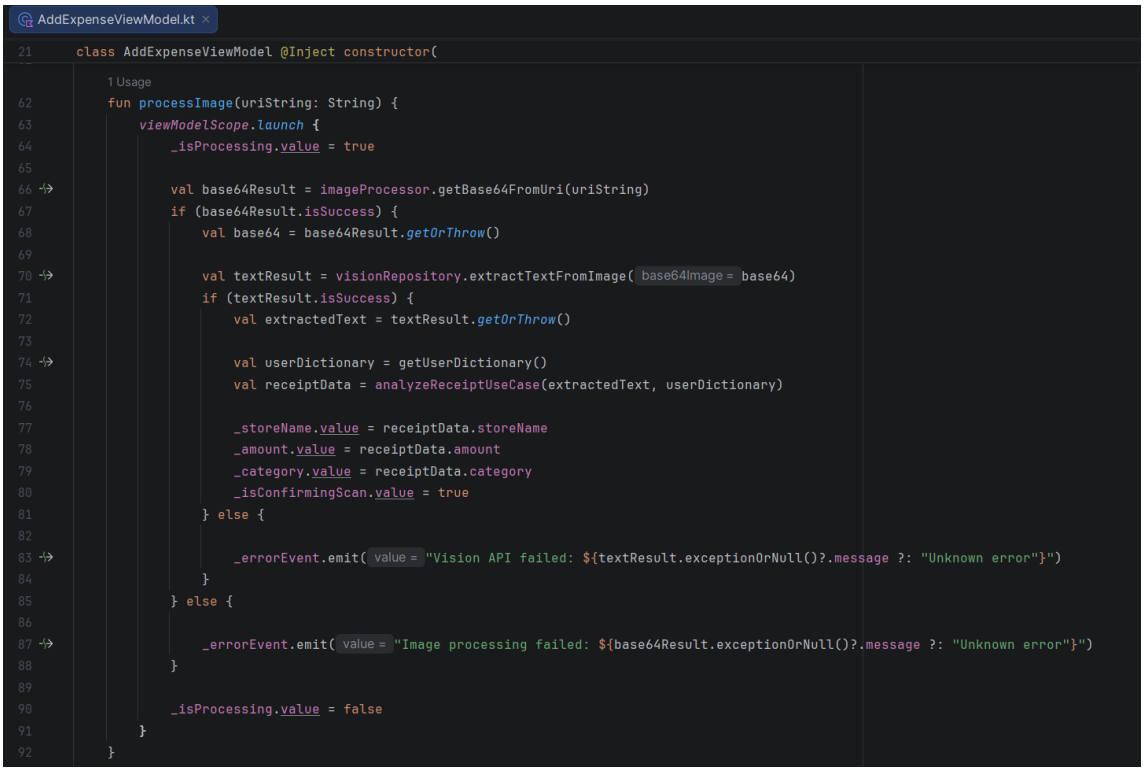
4.4.1 Διαχείριση Κατάστασης (UI State) και ViewModels

Τα ViewModels της εφαρμογής κληρονομούν από την κλάση ViewModel του Android Jetpack. Η διαχείριση της κατάστασης πραγματοποιείται με αντιδραστικό τρόπο (reactive state management) μέσω των ασύγχρονων δομών ροής της βιβλιοθήκης Kotlin Coroutines, ακολουθώντας την εξής αρχιτεκτονική ροή:

- Τα ViewModels διατηρούν εσωτερικά ιδιωτικές (private) ροές μεταβολής τύπου MutableStateFlow, οι οποίες επιτρέπουν την εγγραφή και την επικαιροποίηση των νέων καταστάσεων.
- Για την αποφυγή παραβίασης των επιπέδων αφαίρεσης, οι ροές αυτές εκτίθενται προς τα Composables (Views) ως read-only αντικείμενα τύπου StateFlow, μέσω της συνάρτησης επέκτασης asStateFlow().
- Οι συναρτήσεις Composables παρατηρούν και συλλέγουν τις ροές αυτές μετατρέποντάς τες σε εγγενή κατάσταση της Compose μέσω της μεθόδου collectAsState(), εξασφαλίζοντας ότι οποιαδήποτε τροποποίηση στο State θα προκαλέσει την αυτόματη ανασύνθεση (Recomposition) των επηρεαζόμενων οπτικών στοιχείων.

4.4.1.1 Η ροή εργασίας στο AddExpenseViewModel

Η κλάση AddExpenseViewModel συνιστά χαρακτηριστικό παράδειγμα αυτής της αρχιτεκτονικής, καθώς αναλαμβάνει τον συντονισμό της ασύγχρονης ροής του υποσυστήματος OCR.



Εικόνα 4.24 Κώδικας της processImage του AddExpenseViewModel.kt

Ένα αξιοσημείωτο στοιχείο της επιχειρησιακής λογικής είναι η συνάρτηση getUserDictionary(). Η μέθοδος αυτή ανακτά το ιστορικό των αποθηκευμένων δαπανών από το ExpenseRepository και κατασκευάζει έναν δυναμικό χάρτη συσχέτισης (Map<String, ExpenseCategory>), ο οποίος λειτουργεί ως τοπικό λεξικό καταστημάτων. Έτσι, ο αλγόριθμος προσαρμόζεται στις καταναλωτικές συνήθειες του χρήστη (π.χ. αν ο χρήστης κατηγοριοποιήσει ένα κατάστημα που δεν υπάρχει ήδη στο seedDictionary ως "Shopping", το OCR θα το κατατάξει αυτόματα στην ίδια κατηγορία κατά τις επόμενες σαρώσεις).

4.4.2 Σχεδιασμός Διεπαφής με Jetpack Compose

Η σχεδίαση του γραφικού περιβάλλοντος έγινε με το Jetpack Compose και βασίζεται σε συστατικά στοιχεία του Material Design 3, διασφαλίζοντας οπτική ομοιομορφία σε όλες τις επιμέρους οθόνες της εφαρμογής.

4.4.2.1 Οθόνη Προσθήκης Εξόδου (AddExpenseScreen)

Η συνάρτηση AddExpenseScreen παρέχει μια ευέλικτη διεπαφή που υποστηρίζει τόσο τη χειροκίνητη καταχώριση δαπανών όσο και την αυτοματοποιημένη σάρωση αποδείξεων.

Για τη χειροκίνητη εισαγωγή και τροποποίηση των δεδομένων, χρησιμοποιούνται δομημένα στοιχεία OutlinedTextField (για την επωνυμία και το ποσό) και ένα δυναμικό DropdownMenu για την επιλογή της κατηγορίας δαπάνης.

Όταν ο χρήστης επιλέξει την σάρωση, η οθόνη μεταβαίνει σε περιβάλλον κάμερας. Μόλις ολοκληρωθεί η ασύγχρονη επεξεργασία του OCR και η ροή δεδομένων επιστρέψει τις εξαχθείσες τιμές, ενεργοποιείται μια

κατάσταση επιβεβαίωσης (isConfirmingScan). Η κατάσταση αυτή προβάλλει την οθόνη χειροκίνητης προσθήκης με τα πεδία προσυμπληρωμένα, επιτρέποντας στον χρήστη να επιθεωρήσει, να διορθώσει τυχόν αναγνωστικές αστοχίες και να επικυρώσει τα στοιχεία πριν αυτά εγγραφούν στην τοπική βάση δεδομένων Room.

AddExpenseScreen.kt

```
52    fun AddExpenseScreen(
129       ) { innerPadding →
130           Column(modifier = Modifier.fillMaxSize().padding( paddingValues = innerPadding)) {
143               }
144
145               Box(modifier = Modifier.fillMaxWidth().weight(1f)) {
146                   if (isConfirmingScan) {
147                       ManualExpenseForm(
148                           storeName = storeName,
149                           onStoreNameChange = { addExpenseViewModel.updateStoreName(it) },
150                           amount = amount,
151                           onAmountChange = { addExpenseViewModel.updateAmount( amt = it) },
152                           category = category,
153                           onCategoryChange = { addExpenseViewModel.updateCategory( cat = it) },
154                           currencySymbol = currencyPreference,
155                           onNavigateBack = onNavigateBack,
156                           onSaveExpense = { expenseViewModel.addExpense(it) }
157                       )
158                   } else if (selectedTab == 0) {
159                       CameraUI(
160                           isProcessing = isProcessing,
161                           onImageSelected = { uriString →
162                               addExpenseViewModel.processImage(uriString)
163                           }
164                       )
165                   } else {
166                       ManualExpenseForm(
167                           storeName = storeName,
168                           onStoreNameChange = { addExpenseViewModel.updateStoreName(it) },
169                           amount = amount,
170                           onAmountChange = { addExpenseViewModel.updateAmount( amt = it) },
171                           category = category,
172                           onCategoryChange = { addExpenseViewModel.updateCategory( cat = it) },
173                           currencySymbol = currencyPreference,
174                           onNavigateBack = onNavigateBack,
175                           onSaveExpense = { expenseViewModel.addExpense(it) }
176                       )
177                   }
178               }
179           }
180       }
```

Εικόνα 4.25 Κομμάτι Κώδικα της AddExpenseScreen.kt

4.4.2.2 Ενσωμάτωση Κάμερας με CameraX (CameraPreview)

Η ενσωμάτωση της κάμερας για τη λήψη αποδείξεων βασίζεται στη βιβλιοθήκη CameraX και έχει διαχωριστεί σε δύο εξειδικευμένα γραφικά συστατικά (Composables): το CameraUI και το CameraPreview.

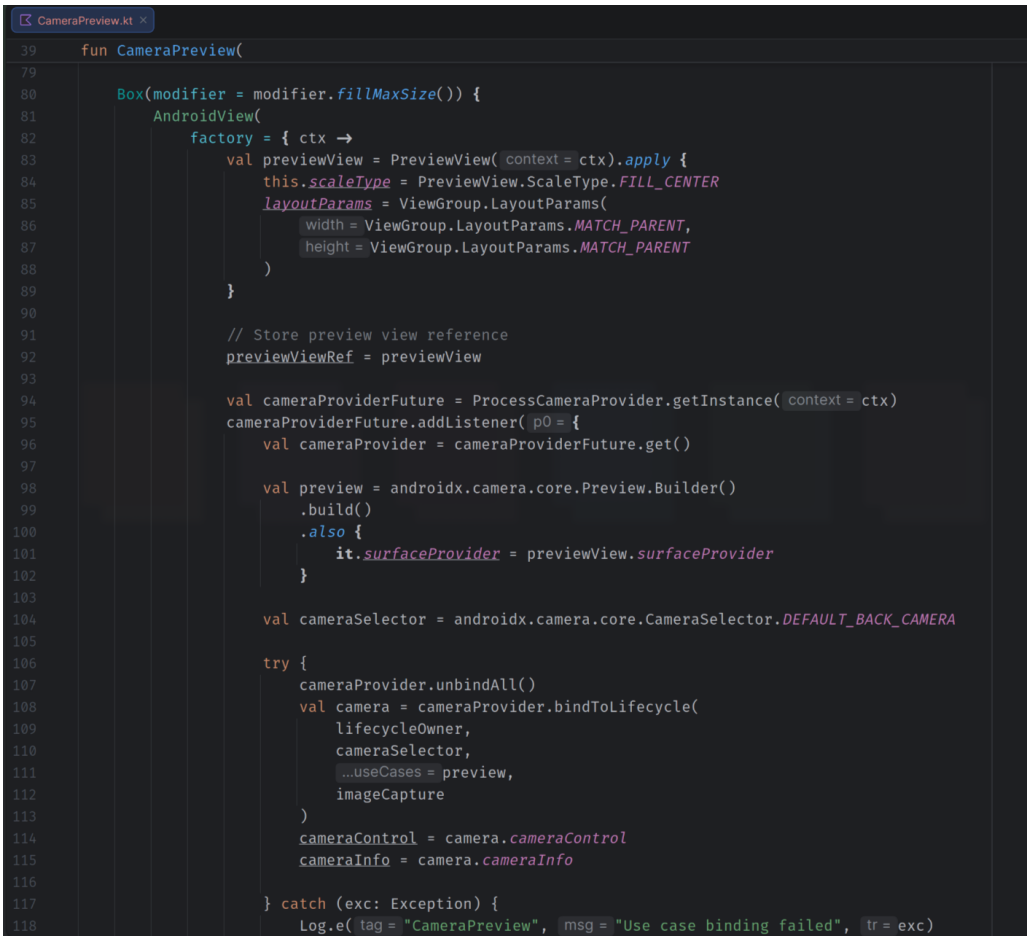
Το CameraUI (components/CameraUI.kt) λειτουργεί ως ο ενορχηστρωτής της κάμερας και είναι υπεύθυνο για:

- **Διαχείριση Αδειών:** Ελέγχει αν έχει παραχωρηθεί η άδεια κάμερας (Manifest.permission.CAMERA) και αν όχι, την αιτείται δυναμικά χρησιμοποιώντας το rememberLauncherForActivityResult.
- **Έλεγχο Σύνδεσης στο Διαδίκτυο:** Καθώς η ανάλυση OCR απαιτεί σύνδεση στο Google Vision API, το CameraUI παρατηρεί την κατάσταση του δικτύου μέσω του rememberIsNetworkAvailable() το οποίο εγγράφει callback στο ConnectivityManager. Αν η συσκευή είναι εκτός σύνδεσης, εμφανίζεται προειδοποιητικό banner στην κορυφή της οθόνης.
- **Λήψη Φωτογραφίας & Ήχο Κλείστρου:** Η συνάρτηση takePhoto() αποθηκεύει την εικόνα προσωρινά στον φάκελο cache της συσκευής και επιστρέφει το τοπικό URI. Κατά τη λήψη αναπαράγεται ο τυπικός ήχος κλείστρου μέσω της κλάσης MediaActionSound για καλύτερη εμπειρία χρήσης.

- **Εναλλακτική Επιλογή από Γκαλερί:** Παρέχεται κουμπί που εκκινεί το PickVisualMedia API του Android για την επιλογή ήδη υπάρχουσας φωτογραφίας από τη βιβλιοθήκη.
- **Overlay Επεξεργασίας:** Όταν η μεταβλητή isProcessing είναι αληθής, εμφανίζεται ένα ημιδιαφανές μαύρο φόντο με CircularProgressIndicator, κλειδώνοντας τα κουμπιά για την αποφυγή διπλών κλήσεων.

Το CameraPreview (components/CameraPreview.kt) αναλαμβάνει την απεικόνιση της κάμερας. Επειδή το Jetpack Compose δεν διαθέτει εγγενές component για την ροή κάμερας, χρησιμοποιείται το AndroidView προκειμένου να ενσωματωθεί το παραδοσιακό PreviewView της CameraX μέσα σε compose layout. Επιπλέον, υλοποιούνται οι παρακάτω λειτουργίες:

- **Αυτόματη Εστίαση με Άγγιγμα (Autofocus-on-Tap):** Μέσω του modifier pointerInput ανιχνεύεται το άγγιγμα του χρήστη στην οθόνη (detectTapGestures). Οι συντεταγμένες του Compose μετατρέπονται σε MeteringPoint μέσω του meteringPointFactory του PreviewView, και εκκινείται η εστίαση μέσω του cameraControl.startFocusAndMetering().
- **Οπτική Ένδειξη Εστίασης:** Στο σημείο αγγίγματος σχεδιάζεται ένας λευκός κύκλος ο οποίος εξασθενεί σταδιακά (fade out) μέσα σε 800ms χρησιμοποιώντας μια Compose Animatable μεταβλητή alpha.
- **Έλεγχος Φλας (Torch):** Μέσω ενός LaunchedEffect που παρατηρεί τη μεταβλητή flashEnabled, ενεργοποιείται ή απενεργοποιείται ο προβολέας (torch) της κάμερας χρησιμοποιώντας τη μέθοδο cameraControl.enableTorch()**.**
- **Αποδέσμευση (Cleanup):** Στο lifecycle event onDispose (μέσω DisposableEffect), εκτελείται η μέθοδος unbindAll() του ProcessCameraProvider για να διασφαλιστεί ότι η κάμερα αποδεσμεύεται αμέσως μόλις ο χρήστης βγει από την οθόνη λήψης, εξοικονομώντας μπαταρία και προστατεύοντας την ιδιωτικότητα.



```kotlin
fun CameraPreview(

    Box(modifier = modifier.fillMaxSize()) {
        AndroidView(
            factory = { ctx ->
                val previewView = PreviewView( context = ctx).apply {
                    this.scaleType = PreviewView.ScaleType.FILL_CENTER
                    layoutParams = ViewGroup.LayoutParams(
                        width = ViewGroup.LayoutParams.MATCH_PARENT,
                        height = ViewGroup.LayoutParams.MATCH_PARENT
                    )
                }

                // Store preview view reference
                previewViewRef = previewView

                val cameraProviderFuture = ProcessCameraProvider.getInstance( context = ctx)
                cameraProviderFuture.addListener( p0 = {
                    val cameraProvider = cameraProviderFuture.get()

                    val preview = androidx.camera.core.Preview.Builder()
                        .build()
                        .also {
                            it.surfaceProvider = previewView.surfaceProvider
                        }

                    val cameraSelector = androidx.camera.core.CameraSelector.DEFAULT_BACK_CAMERA

                    try {
                        cameraProvider.unbindAll()
                        val camera = cameraProvider.bindToLifecycle(
                            lifecycleOwner,
                            cameraSelector,
                            ...useCases = preview,
                            imageCapture
                        )
                        cameraControl = camera.cameraControl
                        cameraInfo = camera.cameraInfo

                    } catch (exc: Exception) {
                        Log.e( tag = "CameraPreview",  msg = "Use case binding failed",  tr = exc)
```

Εικόνα 4.26 Κομμάτι κώδικα του CameraPreview.kt

4.4.2.3 Αρχική Οθόνη και Στατιστικά (HomeScreen & Charts)

Η HomeScreen αποτελεί το κεντρικό dashboard της εφαρμογής και οργανώνεται σε δύο κύριες καρτέλες (Tabs):

Η καρτέλα Μηνιαίας Προβολής ("Τώρα") επικεντρώνεται στα έξοδα του επιλεγμένου μήνα και αποτελείται από:

- **Custom Donut Chart:** Ένα κυκλικό γράφημα δακτυλίου το οποίο έχει σχεδιαστεί αποκλειστικά με το Compose Canvas API. Χρησιμοποιώντας τη μέθοδο drawArc(), ορίζονται διαδοχικά τόξα με διαφορετικά χρώματα για κάθε κατηγορία εξόδων. Οι γωνίες σάρωσης (sweep angles) υπολογίζονται δυναμικά βάσει του ποσοστού συμμετοχής κάθε κατηγορίας στο συνολικό ποσό. Στο κέντρο του δακτυλίου εμφανίζεται δυναμικό κείμενο (π.χ. "Έξοδα" ή "Υπόλοιπο") και το αντίστοιχο ποσό, το οποίο αλλάζει χρώμα σε κόκκινο αν ο χρήστης ξεπεράσει τον συνολικό προϋπολογισμό.
- **Pager Υπο-καρτελών (Sub-tabs):** Κάτω από το γράφημα υπάρχει ένας Pager που επιτρέπει την εναλλαγή μεταξύ:
    a) **Transactions (Συναλλαγές):** Μια λίστα (LazyColumn) με όλες τις καταχωρήσεις του μήνα.
    b) **Limits (Όρια):** Εμφανίζει γραμμές προόδου για τον μηνιαίο προϋπολογισμό καθώς και για κάθε κατηγορία ξεχωριστά (CategoryProgressRow),ώστε ο χρήστης να παρακολουθεί με ευκολία αν πετυχαίνει του οικονομικούς του στόχους.

Η καρτέλα Ετήσιας Προβολής ("Ιστορικό") εμφανίζει τη συνολική εικόνα του έτους και αποτελείται από:

- **Custom Yearly Bar Chart:** Ένα ραβδόγραμμα (YearlyBarChart) το οποίο σχεδιάστηκε επίσης custom με Compose. Κάθε μήνας αναπαρίσταται από μια κάθετη στήλη (Box). Το ύψος της στήλης υπολογίζεται ως κλάσμα (fraction) του μέγιστου ποσού που δαπανήθηκε κατά τη διάρκεια του έτους. Αν τα έξοδα κάποιου μήνα ξεπεράσουν το budget, η αντίστοιχη στήλη χρωματίζεται αυτόματα κόκκινη, προσφέροντας άμεση οπτική πληροφόρηση.
- **Ανάλυση ανά Μήνα:** Κάτω από το ραβδόγραμμα υπάρχει λίστα (Column) με τα συνολικά έξοδα κάθε μήνα. Και εδώ, αν κάποιο σύνολο είναι άνω του ορίου του χρήστη, το ποσό εμφανίζεται κόκκινο.

4.4.2.4 Οθόνη Ρύθμισης Προϋπολογισμού (BudgetSettingsScreen)

Η οθόνη BudgetSettingsScreen επιτρέπει στο χρήστη να διαμορφώσει τον μηνιαίο προϋπολογισμό του. Είναι οργανωμένη σε δύο βασικά τμήματα:

Την Κάρτα Συνολικού Μηνιαίου Προϋπολογισμού η οποία επιτρέπει την εισαγωγή ενός συνολικού ακέραιου ποσού. Χρησιμοποιεί το custom component MinimalistIntegerInput το οποίο εμφανίζει το σύμβολο του προτιμώμενου νομίσματος του χρήστη (π.χ. € μετά τον αριθμό ή $ πριν τον αριθμό) και περιορίζει την είσοδο μόνο σε έγκυρους θετικούς αριθμούς.

Την Κάρτα Ορίων ανά Κατηγορία (Category Limits) η οποία περιέχει:

- **Διπλή Λειτουργία (Toggle Mode):** Ο χρήστης μπορεί να επιλέξει αν θέλει να ρυθμίσει τα όρια ανά κατηγορία ως απόλυτη νομισματική τιμή (π.χ. 200€) ή ως ποσοστό επί του συνολικού budget (π.χ. 20%), πατώντας το κουμπί εναλλαγής στην κορυφή.
- **Δυναμικός Έλεγχος Υπολοίπου:** Στην κορυφή της λίστας εμφανίζεται το υπολειπόμενο ποσό που είναι διαθέσιμο για κατανομή. Ο χρήστης δεν μπορεί να ορίσει όρια κατηγοριών που να ξεπερνούν το συνολικό budget.
- **CategoryLimitRow:** Κάθε κατηγορία εμφανίζεται σε μια γραμμή που περιέχει:
  - **a)** Έναν custom ρυθμιστή Slider για γρήγορη αυξομείωση της τιμής.
  - **b)** Ένα text field MinimalistIntegerInput για ακριβή πληκτρολόγηση.
  - **c)** Μια ετικέτα αυτόματης μετατροπής ακριβώς κάτω από το πεδίο εισαγωγής (π.χ. αν ο χρήστης πληκτρολογεί σε %, εμφανίζεται από κάτω το ισοδύναμο ποσό σε ευρώ, και αντίστροφα).

Με το πάτημα του κουμπιού “Αποθήκευση” ("Save") στην Top App Bar, οι αλλαγές αποθηκεύονται ασύγχρονα στο DataStore μέσω του BudgetViewModel και ο χρήστης επιστρέφει στην αρχική οθόνη.

4.4.2.5 Οθόνες Πρώτης Εισαγωγής (Onboarding Screens)

Η πρώτη επαφή του χρήστη με την εφαρμογή καθοδηγείται από μια σειρά τριών διαδοχικών οθονών καλωσορίσματος και αρχικής ρύθμισης (Onboarding Flow), οι οποίες εμφανίζονται μόνο κατά την πρώτη εκκίνηση της εφαρμογής:

- **OnboardingWelcomeScreen:** Μια οθόνη υποδοχής η οποία καλωσορίζει τον χρήστη και αναφέρει τον σκοπό της εφαρμογής.
- **OnboardingPrefsScreen:** Επιτρέπει στο χρήστη να επιλέξει τις βασικές προτιμήσεις του, όπως τη γλώσσα

της εφαρμογής, το προτιμώμενο νόμισμα και το θέμα της εφαρμογής χωρίς να χρειάζεται να μεταβαίνει στις ρυθμίσεις αργότερα.

- **OnboardingBudgetScreen:** Προτρέπει τον χρήστη να εισάγει το αρχικό μηνιαίο όριο εξόδων (Total monthly budget), ώστε η εφαρμογή να είναι έτοιμη για χρήση αμέσως μετά την ολοκλήρωση της διαδικασίας.

Η κατάσταση ολοκλήρωσης του onboarding αποθηκεύεται ως boolean flag (isOnboardingCompleted) στο DataStore μέσω του UserPreferencesRepository. Κατά την εκκίνηση, το AppNavigation ελέγχει αυτήν την τιμή: αν είναι ψευδής, ο χρήστης κατευθύνεται αυτόματα στην οθόνη καλωσορίσματος, ενώ αν είναι αληθής, μεταφέρεται απευθείας στο κεντρικό Dashboard της εφαρμογής.

4.4.3 Πλοήγηση Εφαρμογής (App Navigation)

Η πλοήγηση και η διαχείριση της στοίβας των οθονών (Backstack) υλοποιείται στο αρχείο AppNavigation.kt με τη χρήση της επίσημης βιβλιοθήκης Compose Navigation. Η λειτουργία του είναι η εξής:

Οι διαθέσιμοι προορισμοί της εφαρμογής ορίζονται με αυστηρή ασφάλεια τύπων μέσω της σφραγισμένης κλάσης (sealed class) Screen.kt, αποτρέποντας σφάλματα runtime.

Χρησιμοποιείται το NavHost για τη δήλωση του γραφήματος πλοήγησης (Navigation Graph), όπου κάθε Composable οθόνη αντιστοιχίζεται σε μια μοναδική συμβολοσειρά διαδρομής (route).

Η μεταφορά δεδομένων και παραμέτρων (π.χ. το ID ενός εξόδου για επεξεργασία) μεταξύ των οθονών εκτελείται μέσω της ενσωμάτωσης ορισμάτων (Arguments) στις διαδρομές πλοήγησης, διασφαλίζοντας την αρχιτεκτονική συνοχή και την ανεξαρτησία των Composables.

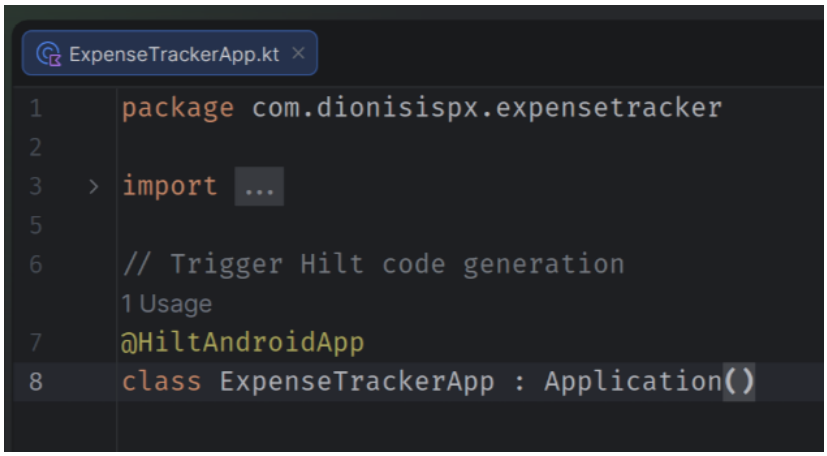
4.4.4 Έγχυση Εξαρτήσεων (Dependency Injection - Dagger Hilt)

Για την αποτελεσματική διαχείριση των εξαρτήσεων μεταξύ των κλάσεων της εφαρμογής και τη διασφάλιση της χαλαρής σύζευξης (loose coupling), χρησιμοποιήθηκε το εξειδικευμένο framework Dagger Hilt, το οποίο αποτελεί την επίσημη αρχιτεκτονική πρόταση της Google για το Android οικοσύστημα.

Η Έγχυση Εξαρτήσεων (Dependency Injection - DI) επιλύει το πρόβλημα της χειροκίνητης αρχικοποίησης και δημιουργίας αντικειμένων. Αντί οι ίδιες οι κλάσεις (για παράδειγμα, τα ViewModels ή τα Use Cases) να αναλαμβάνουν την ευθύνη κατασκευής των αντικειμένων από τα οποία εξαρτώνται (όπως τα Repositories ή οι κλάσεις πρόσβασης στη βάση δεδομένων), οι εξαρτήσεις αυτές "εγχέονται" αυτόματα από το DI framework κατά το χρόνο δημιουργίας και αρχικοποίησης των κλάσεων.

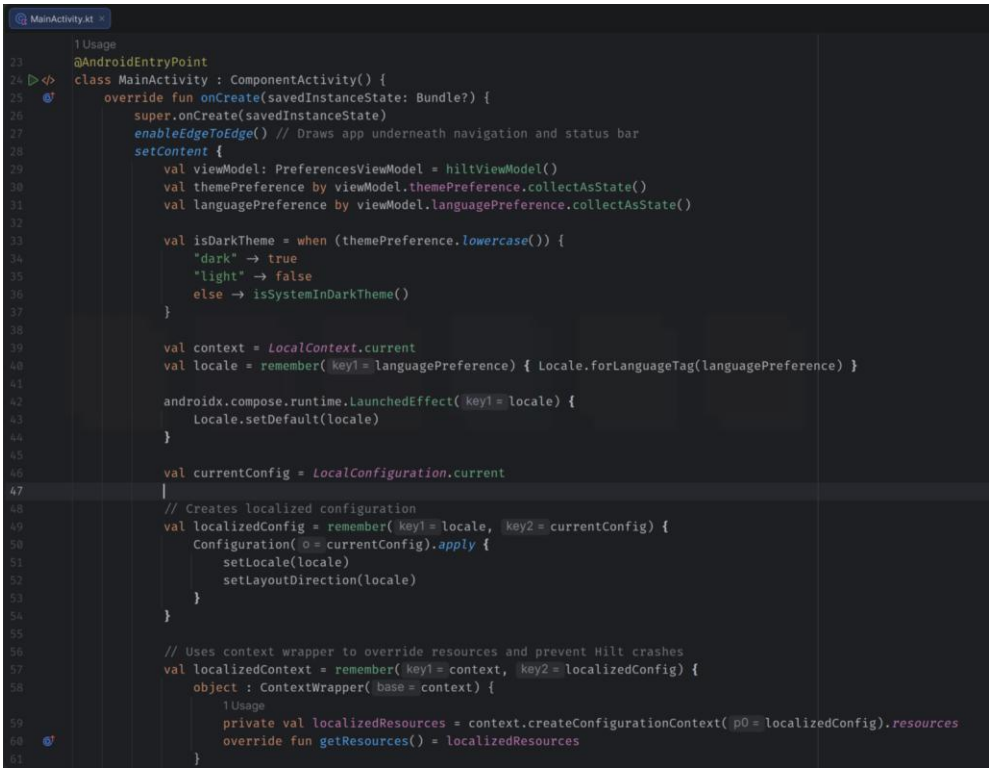
4.4.4.1 Ενεργοποίηση του Hilt στην Εφαρμογή

Η αρχιτεκτονική ενεργοποίηση του Hilt εκκινεί από την κεντρική κλάση της εφαρμογής ExpenseTrackerApp (η οποία κληρονομεί από την κλάση Application του Android framework), φέροντας την annotation @HiltAndroidApp. Η δήλωση αυτή πυροδοτεί την αυτόματη δημιουργία του κώδικα (compile-time code generation) από το Hilt κατά τη διαδικασία της μεταγλώττισης, δημιουργώντας το κεντρικό γράφημα εξαρτήσεων της εφαρμογής:



Εικόνα 4.27 Κώδικας ExpenseTrackerApp.kt

Αντίστοιχα, η κεντρική δραστηριότητα MainActivity επισημαίνεται με την annotation @AndroidEntryPoint. Η σήμανση αυτή επιτρέπει στο Hilt να εγχέει απρόσκοπτα εξαρτήσεις στα Android components (όπως Activities, Fragments και ViewModels), γεφυρώνοντας τον κύκλο ζωής τους με τα αντίστοιχα δοχεία εξαρτήσεων.



Εικόνα 4.28 Κομμάτι Κώδικα της MainActivity.kt

4.4.4.2 Η Μονάδα AppModule (Dagger Hilt Module)

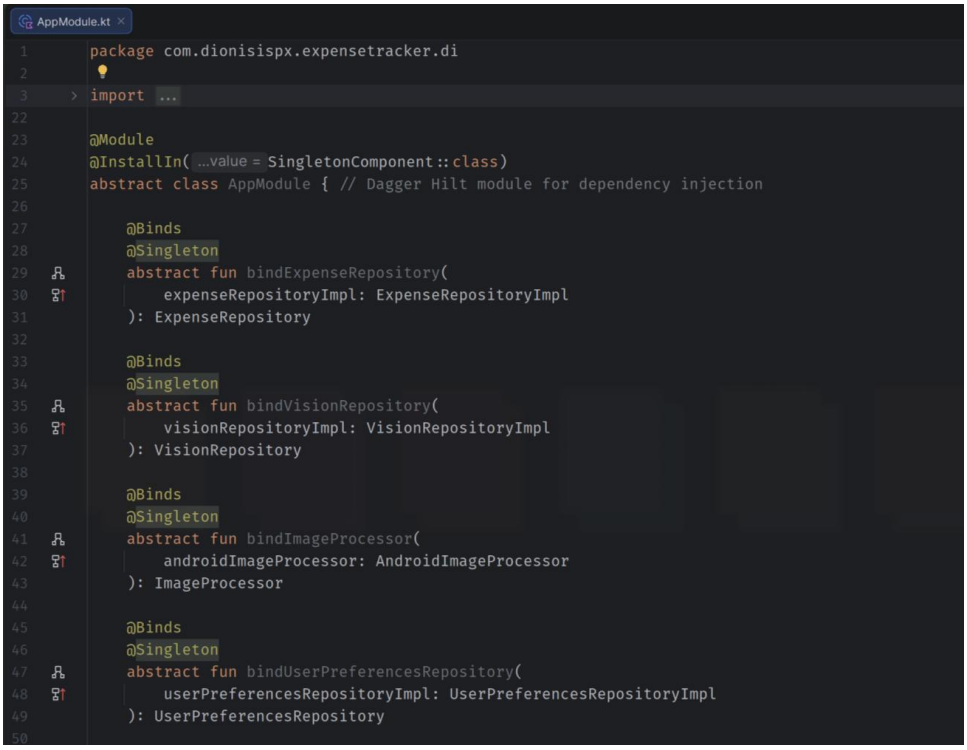
Οι ρυθμίσεις, τα συμβόλαια και οι προδιαγραφές για τη δημιουργία και τη συσχέτιση των εξαρτήσεων ορίζονται στη μονάδα κώδικα AppModule.kt (κάτω από το πακέτο di). Η κλάση αυτή φέρει την annotation @Module και εγκαθίσταται στο SingletonComponent μέσω της δήλωσης @InstallIn(SingletonComponent::class). Αυτό

συνεπάγεται ότι οι εξαρτήσεις που παρέχονται από το συγκεκριμένο module θα έχουν διάρκεια ζωής (lifetime) ίση με τη διάρκεια λειτουργίας ολόκληρης της εφαρμογής, λειτουργώντας ως καθολικά αντικείμενα (Singletons).

Εντός της μονάδας AppModule χρησιμοποιούνται δύο θεμελιώδεις annotations του Dagger, καθεμία από τις οποίες εξυπηρετεί διαφορετικό αρχιτεκτονικό σκοπό:

- **@Binds:** Χρησιμοποιείται για τη σύνδεση μιας αφηρημένης διεπαφής (Interface) με την πραγματική της υλοποίηση (π.χ. τη σύνδεση του ExpenseRepository με την κλάση ExpenseRepositoryImpl). Οι αντίστοιχες μέθοδοι ορίζονται ως αφηρημένες (abstract) για τη βελτιστοποίηση του παραγόμενου κώδικα.
- **@Provides:** Χρησιμοποιείται για την παροχή αντικειμένων που προέρχονται από εξωτερικές βιβλιοθήκες ή απαιτούν σύνθετη παραμετροποίηση και κατασκευαστικά πρότυπα (Builder Pattern) για την αρχικοποίησή τους. Χαρακτηριστικά παραδείγματα αποτελούν η δημιουργία της βάσης δεδομένων Room (provideExpenseDatabase) και η αρχικοποίηση του Retrofit client για την επικοινωνία με το Google Cloud Vision API (provideVisionApi).

Η ενσωμάτωση του Dagger Hilt ολοκληρώνει και θωρακίζει την Clean Architecture δομή της εφαρμογής. Επιτρέπει στα ViewModels να δέχονται Use Cases και στα Use Cases να δέχονται Repositories ως παραμέτρους στους κατασκευαστές τους μέσω της δήλωσης @Inject constructor. Με τον τρόπο αυτό, καμία κλάση δεν διατηρεί γνώση σχετικά με τον τρόπο δημιουργίας των εξαρτήσεών της, διασφαλίζοντας την απόλυτη ανεξαρτησία και την καθαρότητα των αρχιτεκτονικών επιπέδων.



Εικόνα 4.29 Κώδικας AppModule.kt (1)

```
    companion object {
        @Provides
        @Singleton
        fun provideExpenseDatabase(app: Application): ExpenseDatabase {
            return Room.databaseBuilder(
                context = app,
                klass = ExpenseDatabase::class.java,
                name = "expense_db"
            )
            .fallbackToDestructiveMigration() // Fix schema crash
            .build()
        }

        @Provides
        @Singleton
        fun provideExpenseDao(db: ExpenseDatabase): ExpenseDao {
            return db.dao
        }

        @Provides
        @Singleton
        fun provideVisionApi(): VisionApi {
            return retrofit2.Retrofit.Builder()
                .baseUrl("https://vision.googleapis.com/")
                .addConverterFactory(retrofit2.converter.gson.GsonConverterFactory.create())
                .build()
                .create(VisionApi::class.java)
        }
    }
}
```

Εικόνα 4.30 Κώδικας AppModule.kt (2)

5 Δοκιμές, Αξιολόγηση και Εμπειρία Χρήστη

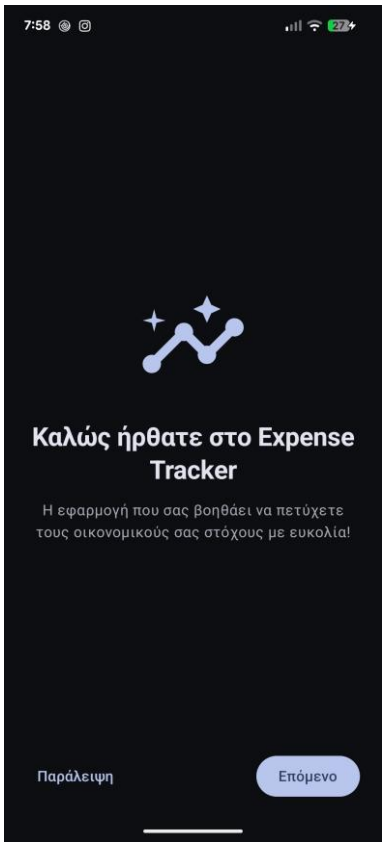
Μετά την ολοκλήρωση της ανάπτυξης της εφαρμογής, πραγματοποιήθηκε μια σειρά δοκιμών προκειμένου να αξιολογηθεί η ορθή λειτουργία της, η ακρίβεια του αλγορίθμου OCR και η συνολική εμπειρία χρήστη (UI/UX). Στο κεφάλαιο αυτό παρουσιάζεται το πλήρες σενάριο χρήσης της εφαρμογής μέσα από στιγμιότυπα οθόνης (screenshots), τα αποτελέσματα των δοκιμών του OCR, καθώς και οι τεχνικές προκλήσεις που επιλύθηκαν κατά την ανάπτυξη.

5.1 Σενάριο Χρήσης και Διεπαφή Χρήστη

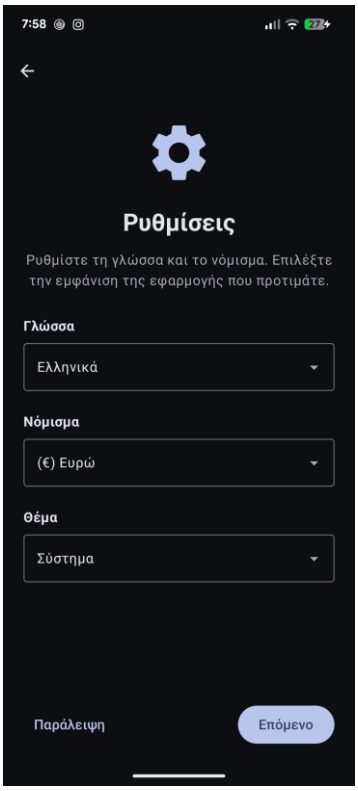
Για την παρουσίαση της διεπαφής και της ροής εργασίας της εφαρμογής, ακολουθεί ένα ολοκληρωμένο σενάριο χρήσης, από την πρώτη εκκίνηση έως την εξαγωγή των δεδομένων:

5.1.1 Αρχική Εκκίνηση και Onboarding

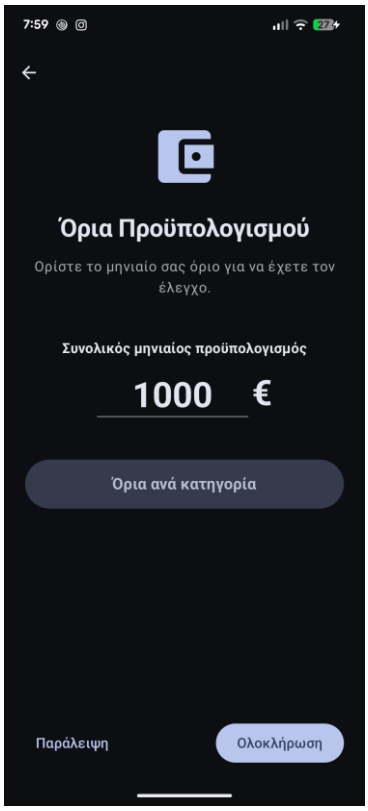
Κατά την πρώτη εκκίνηση της εφαρμογής, ο χρήστης εισέρχεται στη διαδικασία πρώτης παραμετροποίησης (Onboarding) για να επιλέξει τις βασικές του προτιμήσεις και τον επιθυμητό του προϋπολογισμό (αν ο χρήστης επιλέξει να θέσει όρια ανά κατηγορία εμφανίζεται η αντίστοιχη οθόνη που θα παρουσιάσουμε αργότερα).



Εικόνα 5.1 Οθόνη Καλωσορίσματος



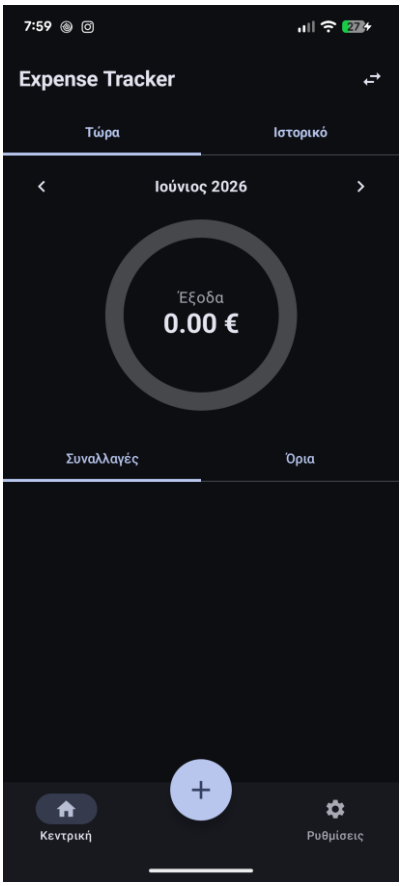
Εικόνα 5.2 Οθόνη Προτιμήσεων Καλωσορίσματος



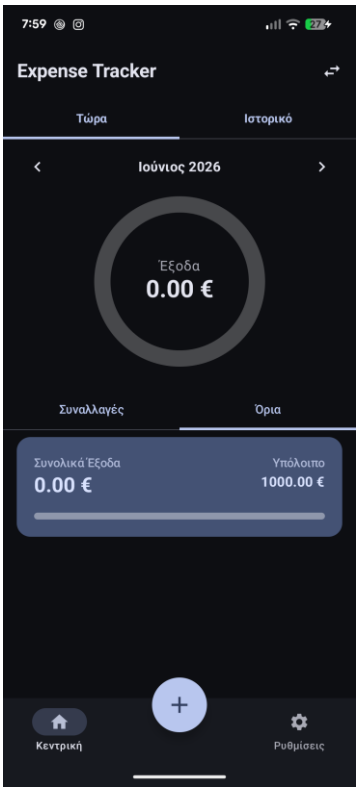
Εικόνα 5.3 Οθόνη Προϋπολογισμού Καλωσορίσματος

5.1.2 Κεντρική Οθόνη σε Κενή Κατάσταση

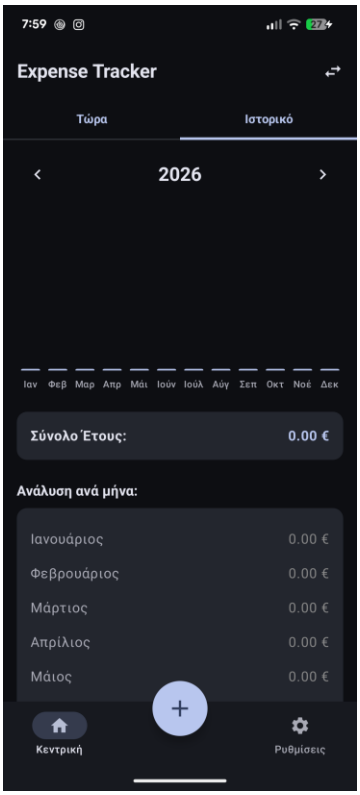
Αφού ολοκληρωθεί το onboarding, ο χρήστης μεταφέρεται στο κεντρικό Dashboard, το οποίο αρχικά δεν περιέχει καταχωρημένα έξοδα.



Εικόνα 5.4 Κεντρική οθόνη - Συναλλάγες (0 έξοδα)



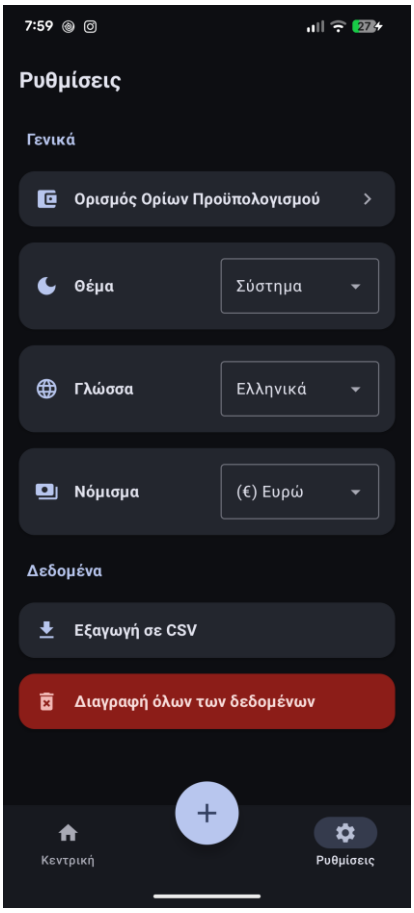
Εικόνα 5.5 Κεντρική οθόνη - Όρια (0 έξοδα)



Εικόνα 5.6 Κεντρική οθόνη - Έτος (0 έξοδα)

5.1.3 Οθόνη Ρυθμίσεων

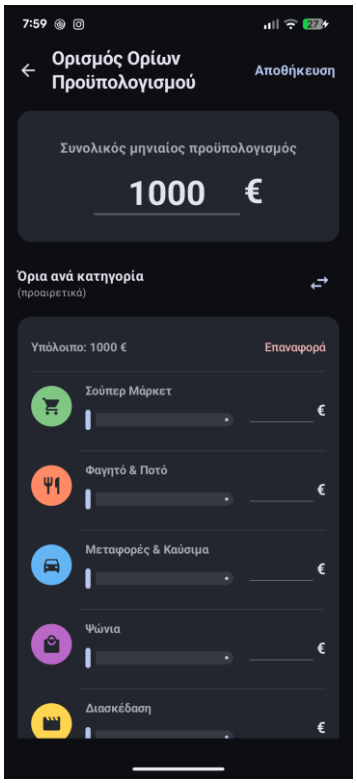
Ο χρήστης μπορεί να μεταβεί στις ρυθμίσεις πατώντας το αντίστοιχο κουμπί για να αλλάξει τη γλώσσα της διεπαφής, το νόμισμα ή το θέμα εμφάνισης. Σε αυτή την οθόνη βρίσκονται και τα κουμπιά για την εξαγωγή και διαγραφή των δεδομένων του.



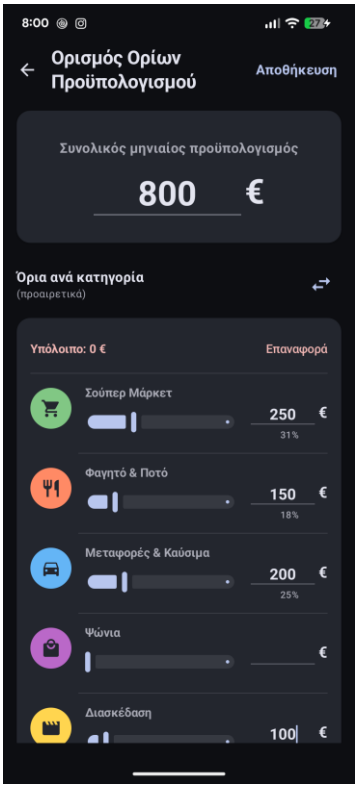
Εικόνα 5.7 Οθόνη Ρυθμίσεων

5.1.4 Ρυθμίσεις Προϋπολογισμού

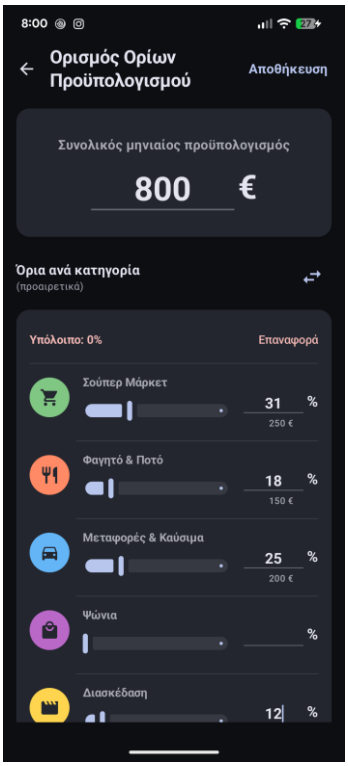
Ο χρήστης μπορεί να ορίσει τα όρια που επιθυμεί στην οθόνη ρύθμισης προϋπολογισμού. Στην ίδια οθόνη, ο χρήστης θέτει επιμέρους όρια για τις κατηγορίες που επιθυμεί (π.χ. Supermarket, Διασκέδαση) χρησιμοποιώντας τους ρυθμιστές (Sliders) είτε σε ευρώ είτε σε ποσοστό. (Η ίδια οθόνη εμφανίζεται και στον onboarding εάν ο χρήστης επιλέξει να θέσει τότε αυτά τα όρια).



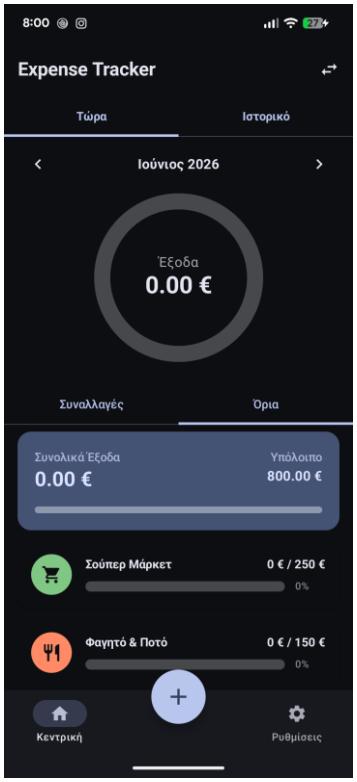
Εικόνα 5.8 Οθόνη Ρύθμισης Προϋπολογισμού με 1000€ μηνιαίο όριο



Εικόνα 5.9 Οθόνη Ρύθμισης Προϋπολογισμού με 800€ μηνιαίο όριο και ρυθμισμένα όρια κατηγοριών (€)



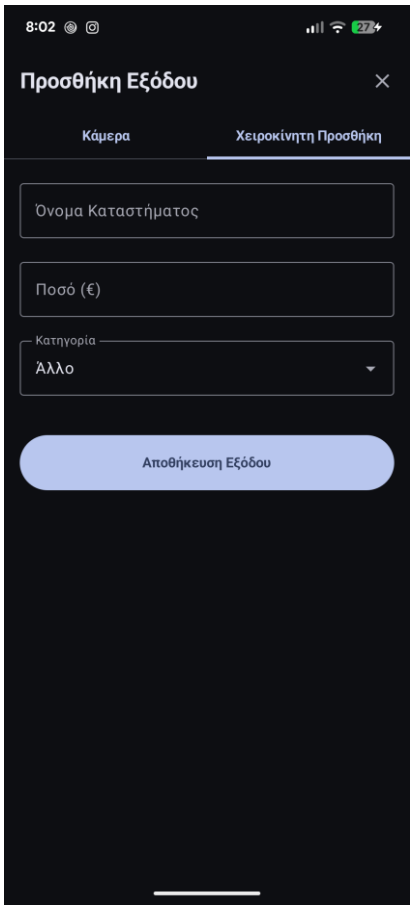
Εικόνα 5.10 Οθόνη Ρύθμισης Προϋπολογισμού με 800€ μηνιαίο όριο και ρυθμισμένα όρια κατηγοριών (%)



Εικόνα 5.11 Κεντρική Οθόνη αφού έχουν ρυθμιστεί όρια κατηγορίας

5.1.5 Χειροκίνητη Προσθήκη Εξόδου

Για έξοδα χωρίς απόδειξη, ο χρήστης χρησιμοποιεί τη φόρμα χειροκίνητης εισαγωγής, συμπληρώνοντας το κατάστημα, το ποσό και την κατηγορία.



Εικόνα 5.12 Κενή Φόρμα Χειροκίνητης Προσθήκης

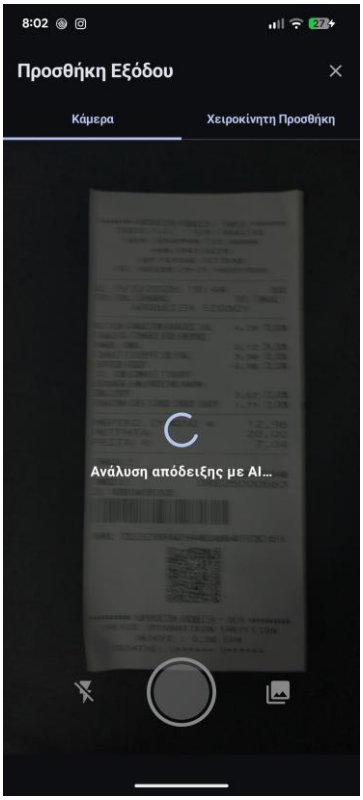
5.1.6 Αυτόματη Προσθήκη με Σάρωση

Ο χρήστης ανοίγει την κάμερα, στοχεύει την απόδειξη και πραγματοποιεί λήψη. Η εικόνα αναλύεται και εμφανίζεται η φόρμα προσθήκης με τα προ-συμπληρωμένα στοιχεία από το OCR.

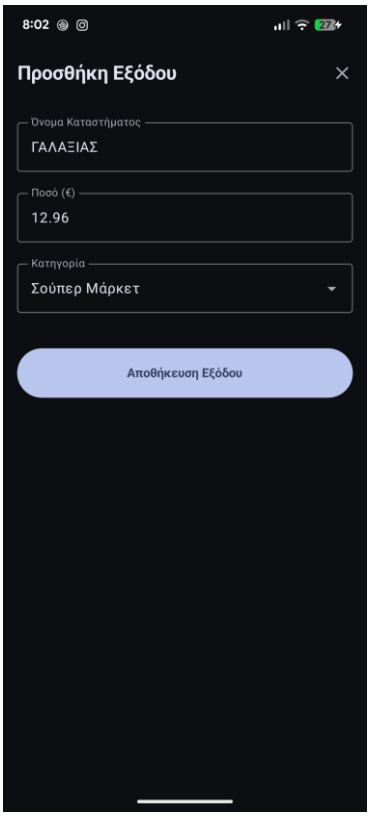
Εικόνα 5.13 Οθόνη Κάμερας (Flash κλειστό)

Εικόνα 5.14 Οθόνη Κάμερας (Flash ανοιχτό)

Εικόνα 5.15 Οθόνη Κάμερας με εστίαση



Εικόνα 5.16 Οθόνη Κάμερας αμέσως μετά την λήψη φωτογραφίας

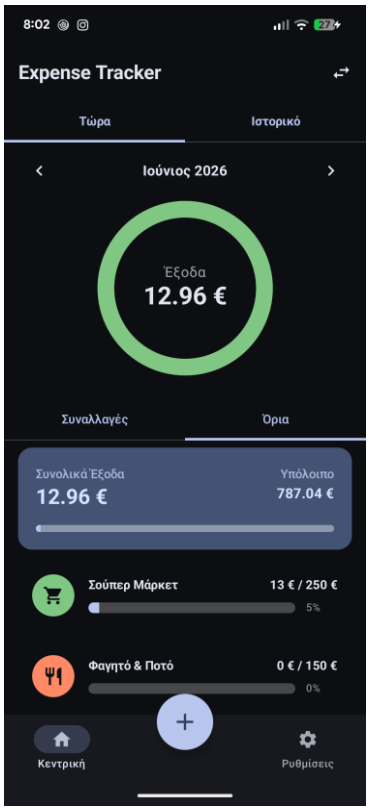


Εικόνα 5.17 Οθόνη Επιβεβαίωσης μετά την λήψη

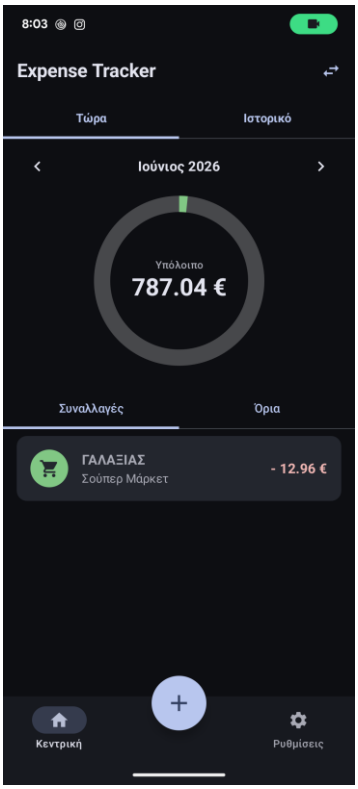
5.1.7 Αρχική Οθόνη μετά τις Πρώτες Προσθήκες

Μετά την αποθήκευση των πρώτων εξόδων, η κεντρική οθόνη ενημερώνεται δυναμικά, εμφανίζοντας τα έξοδα στο Donut Chart και το ιστορικό των συναλλαγών.

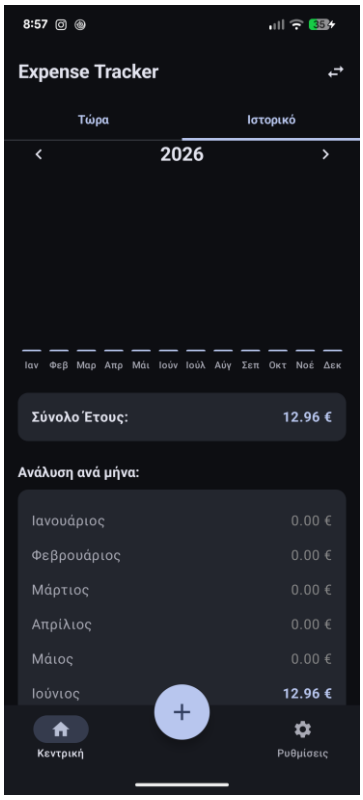
Εικόνα 5.18 Κεντρική οθόνη - Συναλλαγές (1 έξοδο)



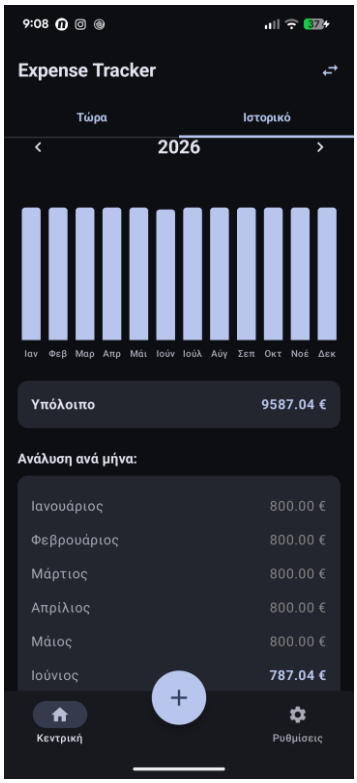
Εικόνα 5.19 Κεντρική οθόνη - Όρια (1 έξοδο)



Εικόνα 5.20 Κεντρική Οθόνη - Συναλλαγές (Υπόλοιπο)



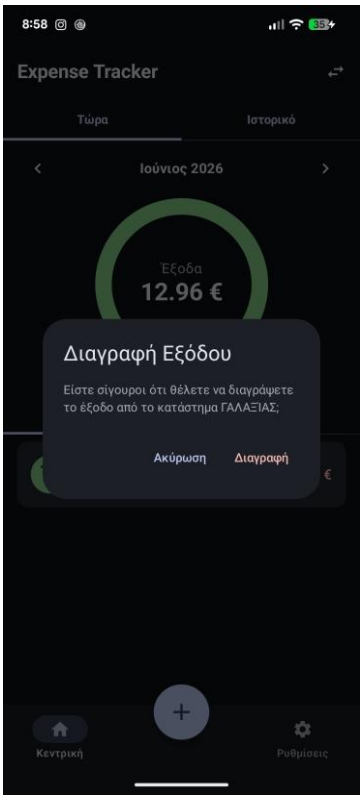
Εικόνα 5.21 Κεντρική Οθόνη - Έτος (1 έξοδο)



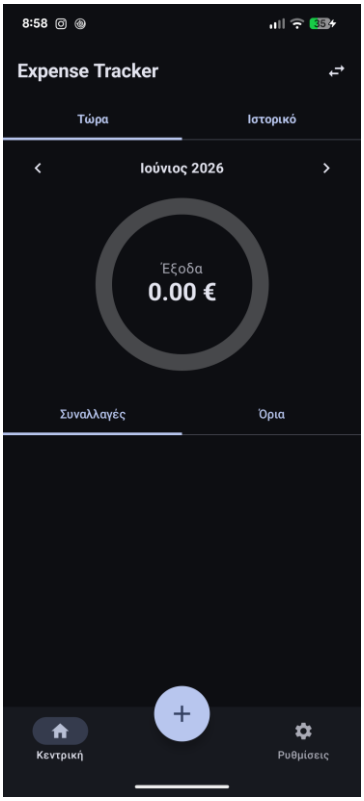
Εικόνα 5.22 Κεντρική Οθόνη - Έτος (Υπόλοιπο)

5.1.8 Διαγραφή Εξόδου

Στην λίστα εξόδων της κεντρική οθόνης, ο χρήστης μπορεί να πατήσει παρατεταμένα πάνω σε ένα έξοδο. Έπειτα από 1 δευτερόλεπτο εμφανίζεται alertDialog ώστε ο χρήστης να επιβεβαιώσει ότι επιθυμεί να διαγράψει το συγκεκριμένο έξοδο. Μετά την επιβαίωση, οι οθόνες ενημερώνονται δυναμικά αφαιρώντας το αντίστοιχο έξοδο.



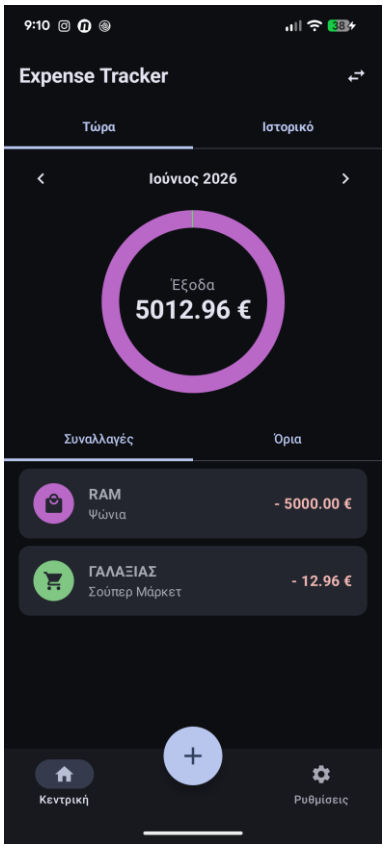
Εικόνα 5.23 Παράθυρο διαλόγου (alertDialog) επιβεβαίωσης διαγραφής εξόδου



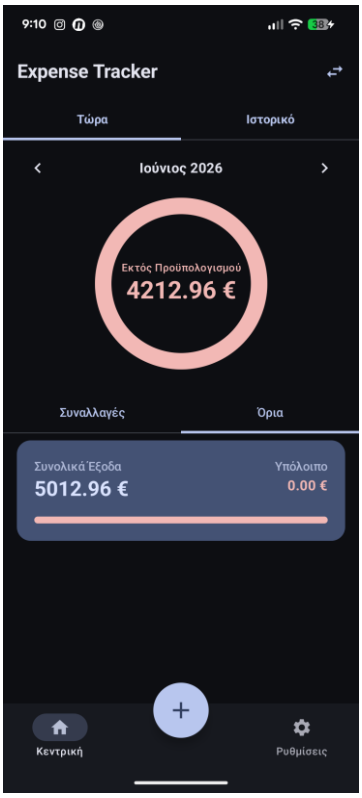
Εικόνα 5.24 Ενημερωμένη κεντρική μετά την διαγραφή

5.1.9 Ξεπέρασμα Ορίων

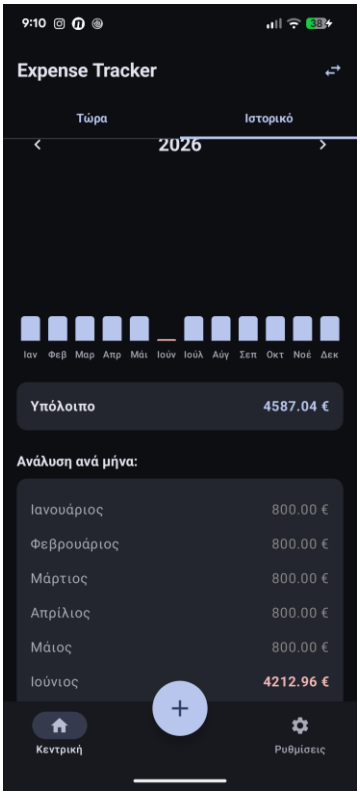
Σε περίπτωση που ο χρήστης ξεπεράσει τα όρια που έχει θέσει, τα αντίστοιχα στοιχεία εμφανίζονται με κόκκινο χρώμα.



Εικόνα 5.25 Κεντρική Οθόνη - Συναλλαγές (εκτός προϋπολογισμού)



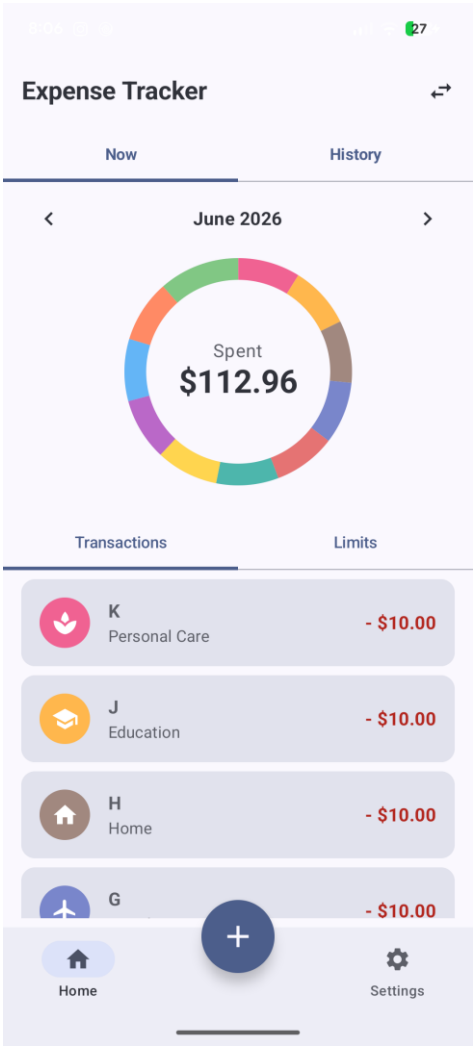
Εικόνα 5.26 Κεντρική Οθόνη - Όρια (εκτός προϋπολογισμού)



Εικόνα 5.27 Κεντρική Οθόνη - Έτος (εκτός προϋπολογισμού)

5.1.10 Παράδειγμα Κεντρικής Οθόνης Χρήστη

Στο παρακάτω στιγμιότυπο φαίνεται η κεντρική οθόνη με ένα έξοδο για κάθε διαθέσιμη κατηγορία. Η εφαρμογή εμφανίζεται στα αγγλικά, με επιλεγμένο νόμισμα το δολάριο και φωτεινό θέμα.



Εικόνα 5.28 Παράδειγμα Κεντρικής Οθόνης

5.1.11 Εξαγωγή δεδομένων

Πατώντας το κουμπί εξαγωγής δεδομένων, η εφαρμογή ρωτάει τον χρήστη που να αποθηκεύσει το αρχείο CSV το οποίο έχει ονομασία “expenses_export_{timestamp}”, όπου {timestamp} η χρονική στιγμή της εξαγωγής. Ανοίγοντας το αρχείο σε έναν επεξεργαστή φυλλαδίων βλέπουμε τα καταστήματα, τα ποσά, τις κατηγορίες και την ημερομηνίες των εξόδων του χρήστη. Τα κελιά και οι κατηγορίες εμφανίζονται στην επιλεγμένη γλώσσα και νόμισμα της εφαρμογής την στιγμή της εξαγωγής (σε αυτή την περίπτωση αγγλικά και δολάριο).

Store	Category	Amount ($)	Date
ΓΑΛΑΞΙΑΣ	Groceries	12.96	6/27/2026 20:02
A	Food & Drink	10.0	6/27/2026 20:03
B	Transport & Fuel	10.0	6/27/2026 20:03
C	Shopping	10.0	6/27/2026 20:03
D	Entertainment	10.0	6/27/2026 20:03
E	Bills & Utilities	10.0	6/27/2026 20:04
F	Health & Fitness	10.0	6/27/2026 20:04
G	Travel	10.0	6/27/2026 20:04
H	Home	10.0	6/27/2026 20:04
J	Education	10.0	6/27/2026 20:04
K	Personal Care	10.0	6/27/2026 20:04

Εικόνα 5.29 Δεδομένα αρχείου εξαγωγής

5.2 Αξιολόγηση της Αναγνώρισης OCR

Η αξιολόγηση της ορθότητας του OCR και των ευρετικών αλγορίθμων ανάλυσης πραγματοποιήθηκε με δύο συμπληρωματικές μεθόδους:

5.2.1 Αυτοματοποιημένες Δοκιμές Pipeline (Unit Tests)

Για τη διασφάλιση της ορθότητας της επιχειρηματικής λογικής (business logic) της εφαρμογής, σχεδιάστηκαν και εκτελέστηκαν αυτοματοποιημένες μονάδες δοκιμών (Unit Tests) χρησιμοποιώντας το framework JUnit 4 (JUnit Team, 2025).

Η δομή της Clean Architecture διευκόλυνε σημαντικά αυτήν τη διαδικασία, καθώς ολόκληρο το Domain Layer (Use Cases, Normalizers) είναι γραμμένο σε καθαρή Kotlin χωρίς καμία εξάρτηση από το Android SDK (όπως Context, View, Lifecycle). Αυτό επέτρεψε την άμεση εκτέλεση των δοκιμών στην JVM του υπολογιστή ανάπτυξης (Host Machine JVM) σε ελάχιστα δευτερόλεπτα, χωρίς την ανάγκη χρήσης εξομοιωτή.

Για να δοκιμαστεί η ανθεκτικότητα των αλγορίθμων μας, τα Unit Tests σχεδιάστηκαν έτσι ώστε να προσομοιώνουν σφάλματα OCR περνώντας "θορυβώδεις" συμβολοσειρές (Noisy Mock Strings) προς ανάλυση:

1. **Δοκιμές Κανονικοποίησης (GreekTextNormalizerTest - 7 τεστ):**
    - Έλεγχος ότι η λέξη με λατινικούς ομόγλυφους χαρακτήρες "SENO" μετατρέπεται επιτυχώς στην καθαρά ελληνική "ΣΕΝΟ".
    - Έλεγχος της μεθόδου Levenshtein Similarity, επαληθεύοντας ότι η ομοιότητα μεταξύ "ΣΚΛΑΒΕΝΙΤΗΣ" και "ΣΚΛΑΒΗΝΙΤΗΣ" (ορθογραφικό λάθος) είναι άνω του 90% (αποδοχή), ενώ με το "ΑΒ ΒΑΣΙΛΟΠΟΥΛΟΣ" είναι κάτω από 30% (απορρίψη).
2. **Δοκιμές Εξαγωγής Τιμών (PriceExtractorTest - 5 τεστ):**
    - **Λατινικά Ομόγλυφα και Κενά:** Δοκιμάστηκε η εξαγωγή ποσού από το string "S Y N O L O 2.50 €" (με λατινικά γράμματα και κενά ενδιάμεσα) και επιβεβαιώθηκε ότι ο αλγόριθμος εξάγει σωστά την τιμή 2.50.
    - **Τιμή σε Επόμενη Γραμμή:** Δοκιμάστηκε η περίπτωση όπου το ποσό βρίσκεται ακριβώς κάτω από τη λέξη "ΣΥΝΟΛΟ".
    - **Δυναμικός Ευρετικός Κανόνας (Cash & Change):** Δοκιμάστηκε η εξαγωγή ποσού όταν δεν

υπάρχει καμία λέξη-κλειδί αλλά οι τιμές ικανοποιούν τη σχέση Total+Change=CashTendered (π.χ. τιμές 12.35, 20.00 και 7.65 στη σειρά, όπου εξάγεται σωστά το 12.35).

- **Εξαίρεση ΦΠΑ:** Επιβεβαιώθηκε ότι ο αλγόριθμος αγνοεί τους συντελεστές ΦΠΑ (6.00, 13.00, 24.00) κατά την αυτόματη επιλογή της υψηλότερης τιμής.

3. **Δοκιμές Ταυτοποίησης Καταστημάτων (StoreNameMatcherTest - 4 τεστ):**

- **Φιλτράρισμα Εταιρικών Καταλήξεων:** Επιβεβαίωση ότι αφαιρούνται σωστά καταλήξεις τύπου "ΙΚΕ", "ΑΕ" κ.λπ., ώστε να γίνεται η σύγκριση μόνο της καθαρής επωνυμίας.
- **Ασαφής Ταυτοποίηση (Fuzzy Match):** Έλεγχος ότι μια επωνυμία με ορθογραφικό/OCR σφάλμα (π.χ. "ΣΚΛΑΒΗΝΙΤΗΣ") ταυτοποιείται επιτυχώς στο σωστό κατάστημα "ΣΚΛΑΒΕΝΙΤΗΣ".
- **Λεξικό Χρήστη (User Dictionary)**: Έλεγχος ότι η εφαρμογή δίνει προτεραιότητα και αντιστοιχεί σωστά τα καταστήματα που έχει εισάγει ο ίδιος ο χρήστης.
- **Εφεδρική Λύση (Fallback Store):** Έλεγχος ότι αν δεν βρεθεί κανένα γνωστό κατάστημα, ο αλγόριθμος επιλέγει ως επωνυμία την πρώτη έγκυρη γραμμή της απόδειξης, παρακάμπτοντας τυποποιημένες φράσεις όπως "ΦΟΡΟΛΟΓΙΚΗ ΑΠΟΔΕΙΞΗ".

Όλες οι δοκιμές (16 στο σύνολό τους) εκτελέστηκαν επιτυχώς (Build Successful) μέσα σε λιγότερο από 5 δευτερόλεπτα, αποδεικνύοντας ότι η επιχειρηματική λογική της εφαρμογής είναι σταθερή και εξαιρετικά ανθεκτική στα τυπικά σφάλματα που εισάγει η διαδικασία ψηφιοποίησης (OCR).

Πίνακας 5.1 Σύνοψη και Κατανόηση Αυτοματοποιημένων Δοκιμών (Unit Tests)

Πακέτο Δοκιμών (Test Class)	Πλήθος Τεστ	Κύριο Αντικείμενο Ελέγχου	Αποτέλεσμα (Status)
GreekTextNormalizerTest	7	Αφαίρεση τόνων, μετατροπή πεζών/κεφαλαίων και εξάλειψη λατινικών ομόγλυφων χαρακτήρων.	Επιτυχία (Success)
PriceExtractorTest	5	Εντοπισμός αριθμών κοντά σε λέξεις-κλειδιά, ευρετικός κανόνες Cash/Change και εξαίρεση συντελεστών ΦΠΑ.	Επιτυχία (Success)
StoreNameMatcherTest	4	Φιλτράρισμα εταιρικών καταλήξεων (ΑΕ, ΙΚΕ), ασαφής ταυτοποίηση Levenshtein και προτεραιότητα λεξικού χρήστη.	Επιτυχία (Success)
Σύνολο Δοκιμών	16	Έλεγχος ανθεκτικότητας του Receipt OCR Pipeline	16 / 16 Περασμένα (Passed)

5.2.2 Δοκιμές σε Πραγματικές Αποδείξεις

Φορτώντας την εφαρμογή σε πραγματικό κινητό τηλέφωνο, είχαμε την δυνατότητα να την δοκιμάσουμε στην πράξη σε περίπου 20 διαφορετικές αποδείξεις. Το δείγμα μας είχε μεγάλη ποικιλία ώστε να καλύπτει ένα ευρύ φάσμα αποδείξεων. Αυτές μπορούν να κατηγοριοποιηθούν ως εξής:

- **Εύκολες αποδείξεις:** Καθαρή εκτύπωση, οριζόντια τοποθέτηση και ιδανικός φωτισμός.
- **Μεγάλες αποδείξεις:** Αποδείξεις σούπερ μάρκετ με πολλά προϊόντα, όπου το τελικό σύνολο βρισκόταν πολύ χαμηλά στο κείμενο.
- **Τσαλακωμένες/Αλλοιωμένες αποδείξεις:** Αποδείξεις με έντονες πτυχώσεις, σκιές ή μερικώς ξεθωριασμένο μελάνι.

Εικόνα 5.30 Δείγμα Πραγματικών Αποδείξεων

Καθώς υπάρχουν πολλοί παράγοντες που μπορούν να επηρεάσουν τα αποτελέσματα του OCR, είναι δύσκολο να δοθεί ακριβές ποσοστό επιτυχίας. Παρά το γεγονός αυτό, τα αποτελέσματα έδειξαν ότι ο αλγόριθμός μας μπορεί να εντοπίσει με ακρίβεια τόσο το ποσό πληρωμής, όσο και το όνομα του καταστήματος, εφόσον αυτά είναι εμφανή στην φωτογραφία. Σε αντίξοες συνθήκες ενδέχεται να χρειαστούν παραπάνω από μία λήψεις.

5.3 Επίλυση Προβλημάτων κατά την Ανάπτυξη

Κατά τη διάρκεια του κύκλου ανάπτυξης της εφαρμογής, ανακύψαν σημαντικά τεχνικά προβλήματα, κυρίως γύρω από τη διαδικασία του OCR, τα οποία επιλύθηκαν διαδοχικά:

5.3.1 Επιλογή Μηχανής OCR (ML Kit, Tesseract, Cloud Vision)

Η μεγαλύτερη πρόκληση της εργασίας ήταν η επιλογή της κατάλληλης μηχανής OCR για την αναγνώριση ελληνικών χαρακτήρων:

- **Google ML Kit (Text Recognition):** Αρχικά χρησιμοποιήθηκε το ML Kit της Google για on-device επεξεργασία. Ωστόσο, η έκδοση που υποστήριζε ελληνικά εμφάνισε χαμηλή ακρίβεια σε θερμικό χαρτί

με μικρές γραμματοσειρές και αλλοιώσεις.

- **Tesseract OCR (Local):** Στη συνέχεια δοκιμάστηκε η ενσωμάτωση της ανοικτής βιβλιοθήκης Tesseract τοπικά στη συσκευή. Τα αποτελέσματα ήταν εξαιρετικά απογοητευτικά, καθώς η αναγνώριση των ελληνικών χαρακτήρων παρήγαγε πολύ θόρυβο (noisy text), ειδικά σε φωτογραφίες ληφθείσες με κάμερα κινητού σε μη ιδανικές συνθήκες.
- **Google Cloud Vision API:** Τελικά, επιλέχθηκε το Cloud Vision API της Google. Παρόλο που απαιτεί σύνδεση στο διαδίκτυο, τα αποτελέσματά του ήταν κλάσεις ανώτερα, παρέχοντας το πιο καθαρό και δομημένο κείμενο, επιτρέποντας στους δικούς μας parsing αλγορίθμους να λειτουργήσουν με υψηλή ακρίβεια.

5.3.2 Βελτιστοποίηση Αποστολής Εικόνας (Payload & Rotation)

Κατά τη χρήση του Cloud Vision API, προέκυψαν δύο σοβαρά προβλήματα:

- **Μέγεθος Αρχείου:** Η CameraX παρήγαγε φωτογραφίες 5-10MB, προκαλώντας καθυστερήσεις και timeouts. Η λύση δόθηκε στον AndroidImageProcessor συμπιέζοντας την εικόνα στα 1024px (JPEG 80%), μειώνοντας το μέγεθος στα ~150KB (98% μείωση) και το χρόνο απόκρισης στα 2 δευτερόλεπτα.
- **Προσανατολισμός (Rotation):** Πολλές συσκευές έστελναν τη φωτογραφία περιστραμμένη κατά 90 μοίρες λόγω του τρόπου που αποθηκεύει τα pixel ο αισθητήρας. Το πρόβλημα λύθηκε διαβάζοντας τα EXIF metadata μέσω της ExifInterface και εφαρμόζοντας Matrix transformation για τη χειροκίνητη περιστροφή του Bitmap πριν την αποστολή.

5.3.3 Προκλήσεις στον Εντοπισμό του Συνόλου και του Καταστήματος

Η εξαγωγή των πληροφοριών από το raw κείμενο εμφάνισε δυσκολίες, συγκεκριμένα:

- **Εντοπισμός Συνόλου:** Πολλές αποδείξεις περιείχαν πολλαπλά ποσά (μερικά σύνολα, αξία ΦΠΑ, μετρητά, ρέστα). Ο PriceExtractor βελτιώθηκε ώστε να εξαιρεί τα standard ποσοστά ΦΠΑ (6, 13, 24) και να χρησιμοποιεί μαθηματικές σχέσεις (π.χ. Total+Change=Cash) για να ξεχωρίζει το σωστό ποσό.
- **Μπερδέματα με Εταιρικές Καταλήξεις:** Κατά την ταυτοποίηση των καταστημάτων, η παρουσία νομικών καταλήξεων (όπως "ΙΚΕ", "ΑΕ", "ΕΠΕ") “μπέρδευε” τις συγκρίσεις Levenshtein. Το πρόβλημα λύθηκε στον StoreNameMatcher φιλτράροντας αυτές τις λέξεις-κλειδιά (companySuffixes) πριν τη σύγκριση, ενώ ο GreekTextNormalizer ομογενοποίησε τους λατινικούς ομόγλυφους χαρακτήρες.

5.3.4 Δυσκολίες στη Διεπαφή Χρήστη (UI)

Η ανάπτυξη της διεπαφής με το Jetpack Compose ήταν εξαιρετικά ομαλή λόγω της δηλωτικής (declarative) φύσης του framework, η οποία απέτρεψε τα κλασικά bugs διαχείρισης κατάστασης (state management) του παλαιού View System.

Η μοναδική δυσκολία αφορούσε την ενσωμάτωση του camera preview της CameraX (που είναι View-based) μέσα στο Compose. Αυτό επιλύθηκε καθαρά με τη χρήση του AndroidView factory και τη σωστή σύνδεση της κάμερας στο lifecycle του Compose Composable, διασφαλίζοντας ότι η κάμερα αποδεσμεύεται (unbindAll()) αμέσως μόλις ο χρήστης βγει από την οθόνη.

6 Συμπεράσματα, Περιορισμοί και Μελλοντικές Βελτιώσεις

Στο τελικό αυτό κεφάλαιο παρουσιάζεται μια συνολική αποτίμηση των αποτελεσμάτων της πτυχιακής εργασίας, αναλύονται οι τρέχοντες αρχιτεκτονικοί και λειτουργικοί περιορισμοί του συστήματος και προτείνονται συγκεκριμένες μελλοντικές επεκτάσεις για την περαιτέρω αναβάθμιση της εφαρμογής.

6.1 Συμπεράσματα

Αντικείμενο της παρούσας πτυχιακής εργασίας ήταν ο σχεδιασμός, η αρχιτεκτονική και η υλοποίηση μιας έξυπνης Android εφαρμογής διαχείρισης προσωπικών εξόδων (“Expense Tracker”), η οποία ενσωματώνει τεχνολογία Οπτικής Αναγνώρισης Χαρακτήρων (OCR) για την αυτοματοποιημένη καταχώρηση αποδείξεων.

Η εφαρμογή αναπτύχθηκε σύμφωνα με τα πλέον σύγχρονα βιομηχανικά πρότυπα ανάπτυξης λογισμικού:

- **Αρχιτεκτονική Δομή (Clean Architecture & MVVM):** Η υιοθέτηση των προτύπων αυτών διασφάλισε τον πλήρη διαχωρισμό των ευθυνών (Separation of Concerns) και την απομόνωση της επιχειρησιακής λογικής από το γραφικό περιβάλλον και το Android Framework. Το γεγονός αυτό κατέστησε τον κώδικα εξαιρετικά ελέγξιμο μέσω αυτοματοποιημένων δοκιμών (testable), εύκολα συντηρήσιμο και έτοιμο για μελλοντικές επεκτάσεις.
- **Διεπαφή Χρήστη (Jetpack Compose & Material Design 3):** Προσέφερε ένα σύγχρονο, δυναμικό και πλήρως αποκρινόμενο (responsive) περιβάλλον εργασίας, το οποίο ενσωματώνει εγγενώς τη σκοτεινή λειτουργία (Dark Theme) και παρέχει οπτικοποίηση των στατιστικών δεδομένων μέσω ειδικά προσαρμοσμένων διαγραμμάτων (Donut και Bar Charts).
- **Μηχανική Όραση (CameraX & Google Vision API):** Προσέφερε μια ομαλή ροή λήψης φωτογραφίας με λειτουργίες εστίασης και φακού, και ταχύτατη εξαγωγή των απαιτούμενων δεδομένων με τη χρήση εξειδικευμένων ευρετικών αλγορίθμων (Regular Expressions, Levenshtein Distance, N-Grams) που αναπτύχθηκαν για την αντιμετώπιση του θορύβου του OCR.

Η πρακτική αξιολόγηση της εφαρμογής επιβεβαίωσε ότι το σύστημα εκπληρώνει πλήρως τους αντικειμενικούς του σκοπούς. Μειώνει δραστικά τον απαιτούμενο χρόνο για τη συστηματική καταγραφή των καθημερινών δαπανών, ελαχιστοποιώντας την ανάγκη για χειροκίνητη πληκτρολόγηση. Η εξάλειψη των εμποδίων εισόδου ενθαρρύνει τον χρήστη να τηρεί με συνέπεια το χρηματοοικονομικό του πλάνο, βοηθώντας τον να παραμένει εντός των καθορισμένων ορίων του προϋπολογισμού του και να αποκτήσει σαφή επίγνωση των καταναλωτικών του συνηθειών.

6.2 Περιορισμοί του Συστήματος

Παρά την επιτυχή υλοποίηση, το σύστημα παρουσιάζει ορισμένους περιορισμούς που υπαγορεύονται από τις τεχνολογικές επιλογές:

- **Απαίτηση Σύνδεσης στο Διαδίκτυο (Cloud-Based OCR):** Η επεξεργασία της εικόνας και η εξαγωγή του κειμένου βασίζονται αποκλειστικά στο Google Cloud Vision API. Αν ο χρήστης δεν έχει πρόσβαση στο διαδίκτυο (Wi-Fi ή δεδομένα κινητής), η λειτουργία της κάμερας δεν είναι εφικτή και η καταχώρηση χρειάζεται να γίνει χειροκίνητα.
- **Εξάρτηση από την Φυσική Κατάσταση του Εγγράφου:** Η ακρίβεια του αλγορίθμου OCR επηρεάζεται

άμεσα από εξωτερικούς παράγοντες, όπως η ένταση του φωτισμού κατά τη λήψη, η ύπαρξη τσακίσεων στο χαρτί ή το ξεθωριασμένο μελάνι των θερμικών εκτυπώσεων. Σε εξαιρετικά θορυβώδεις εισόδους, ο δείκτης ευστοχίας μειώνεται, απαιτώντας τη χειροκίνητη διόρθωση των στοιχείων από τον χρήστη στη φόρμα επιβεβαίωσης.

- **Περιορισμένος Όγκος Δειγμάτων Δοκιμής (Evaluation Sample Size):** Παρά το γεγονός ότι ο αλγόριθμος OCR και οι ευρετικοί κανόνες εξαγωγής στοιχείων αξιολογήθηκαν επιτυχώς με τη χρήση πραγματικών αποδείξεων (περίπου 20 δειγμάτων από διαφορετικούς εμπορικούς κλάδους), το μέγεθος αυτού του συνόλου δεδομένων δοκιμής παραμένει περιορισμένο. Ως εκ τούτου, δεν μπορεί να διασφαλιστεί με απόλυτη στατιστική βεβαιότητα η ίδια υψηλή ακρίβεια σε ευρύτερης κλίμακας δοκιμές ή σε αποδείξεις με ιδιόμορφες δομές μορφοποίησης κειμένου που δεν συμπεριλήφθηκαν στο δείγμα ελέγχου.
- **Τοπική Αποθήκευση Δεδομένων (Single-Device Storage):** Για λόγους προστασίας της ιδιωτικότητας και αρχιτεκτονικής απλότητας, η αποθήκευση εκτελείται αποκλειστικά στην τοπική βάση δεδομένων της συσκευής (Room Database / DataStore). Ως εκ τούτου, δεν υποστηρίζεται ο συγχρονισμός των δεδομένων μεταξύ πολλαπλών συσκευών του ίδιου χρήστη, ενώ σε περίπτωση απώλειας ή αντικατάστασης της συσκευής, η μεταφορά του ιστορικού περιορίζεται από την τρέχουσα απουσία μηχανισμού εισαγωγής δεδομένων από εξωτερικές πηγές.

6.3 Μελλοντικές Βελτιώσεις

Με σκοπό την εξέλιξη της εφαρμογής σε ένα ολοκληρωμένο και ανταγωνιστικό πληροφοριακό σύστημα, προτείνονται οι ακόλουθες μελλοντικές επεκτάσεις:

6.3.1 Υλοποίηση Τοπικού Μοντέλου Οπτικής Αναγνώρισης (On-Device OCR)

Για την άρση της εξάρτησης από το διαδίκτυο, προτείνεται η ενσωμάτωση ενός τοπικού μηχανισμού OCR που θα εκτελείται αποκλειστικά στους υπολογιστικούς πόρους της συσκευής. Η υλοποίηση αυτή παρουσιάζει υψηλό βαθμό δυσκολίας, καθώς οι έτοιμες on-device λύσεις (π.χ. Google ML Kit) στερούνται εγγενούς, υψηλής ακρίβειας υποστήριξης για την ελληνική γλώσσα σε δομημένα έγγραφα, ενώ ανοιχτές βιβλιοθήκες (π.χ.Tesseract OCR) απαιτούν εκτενή προεπεξεργασία εικόνας και εξειδικευμένη εκπαίδευση μοντέλων προκειμένου να αποδώσουν σε γραμματοσειρές αποδείξεων. Η προσθήκη ενός τέτοιου μοντέλου θα λειτουργούσε ως εφεδρικός μηχανισμός (Fallback), επιτρέποντας την offline χρήση της κάμερας, με αποδεκτή ωστόσο μείωση της ακρίβειας σε σχέση με την υπολογιστική ισχύ των μοντέλων νέφους της Google.

6.3.2 Προαιρετικός Συγχρονισμός στο Υπολογιστικό Νέφος (Optional Cloud Synchronization)

Για την εξάλειψη του κινδύνου απώλειας δεδομένων, προτείνεται η ενσωμάτωση μιας cloud πλατφόρμας, όπως το Google Firebase. Η δημιουργία λογαριασμού (Sign-Up) θα είναι εντελώς προαιρετική, διατηρώντας την αρχική φιλοσοφία της εφαρμογής για άμεση χρήση χωρίς εμπόδια (frictionless onboarding) και ιδιωτικότητα των δεδομένων. Οι χρήστες που θα επιλέγουν τη συγκεκριμένη υπηρεσία θα αποκτούν τη δυνατότητα αυτόματου συγχρονισμού των δαπανών τους σε πραγματικό χρόνο μεταξύ πολλαπλών συσκευών.

6.3.3 Διασύνδεση με Τραπεζικά APIs (PSD2 Open Banking)

Μια εξαιρετικά χρήσιμη επέκταση θα ήταν η διασύνδεση της εφαρμογής με τις ελληνικές και ευρωπαϊκές τράπεζες μέσω των ανοικτών τραπεζικών διεπαφών (PSD2 APIs). Κατόπιν ρητής έγκρισης και αυθεντικοποίησης από τον χρήστη, η εφαρμογή θα δύναται να ανακτά αυτόματα συναλλαγές που εκτελούνται μέσω χρεωστικών/πιστωτικών καρτών ή ηλεκτρονικής τραπεζικής (e-banking). Η λειτουργία αυτή θα επιτρέπει την ακριβιαία κατηγοριοποίηση των εξόδων, καταργώντας ακόμη και την ανάγκη λήψης φωτογραφίας της φυσικής απόδειξης.

6.3.4 Έξυπνες Προβλέψεις Προϋπολογισμού (AI-Driven Budgeting)

Η ενσωμάτωση τοπικών μοντέλων Μηχανικής Μάθησης (Machine Learning), προσανατολισμένων στην ανάλυση χρονοσειρών, θα επιτρέψει τη μετάβαση από την απλή καταγραφή στην προγνωστική ανάλυση οικονομικών δεδομένων. Το υποσύστημα αυτό θα προσφέρει:

- **Προληπτικές Ειδοποιήσεις (Predictive Alerts):** Μέσω της ανάλυσης του τρέχοντος ρυθμού κατανάλωσης, το σύστημα θα προειδοποιεί έγκαιρα τον χρήστη (π.χ. “Βάσει των καταναλωτικών τάσεων, προβλέπεται υπέρβαση του ορίου της κατηγορίας 'Ψώνια' εντός των επόμενων 5 ημερών”).
- **Αυτοματοποιημένη Βαθμονόμηση Ορίων:** Δυναμική εισήγηση του βέλτιστου μηνιαίου προϋπολογισμού, λαμβάνοντας υπόψη τα σταθερά έσοδα, τα πάγια μηνιαία έξοδα και τις αποκλίσεις των προηγούμενων περιόδων.

6.3.5 Ανάπτυξη Γραφικών Στοιχείων Αρχικής Οθόνης (Home Screen Widgets)

Για την περαιτέρω επιτάχυνση της διαδικασίας καταχώρισης, προτείνεται η ανάπτυξη Android Widgets για την αρχική οθόνη της συσκευής μέσω της σύγχρονης δηλωτικής βιβλιοθήκης Jetpack Glance. Τα στοιχεία αυτά θα προσφέρουν άμεση οπτική πληροφόρηση σχετικά με το υπόλοιπο του μηνιαίου προϋπολογισμού, καθώς και μια συντόμευση άμεσης προσθήκης, η οποία θα ενεργοποιεί το περιβάλλον της κάμερας για τη σάρωση μιας απόδειξης με ένα μόνο άγγιγμα.

6.3.6 Εισαγωγή Δεδομένων από CSV (CSV Import)

Σε συνέχεια του υπάρχοντος μηχανισμού εξαγωγής, η υλοποίηση της αντίστροφης λειτουργίας εισαγωγής δεδομένων (CSV Import) κρίνεται ως ιδιαίτερα ωφέλιμη. Η δυνατότητα αυτή θα επιτρέψει την εύκολη μετάβαση χρηστών από άλλα πληροφοριακά συστήματα, καθώς και την απευθείας ενσωμάτωση μηνιαίων αναφορών κίνησης λογαριασμών (bank statements) από το e-banking.

Άλλο ένα αρχιτεκτονικό πλεονέκτημα αυτής της προσέγγισης είναι ότι προσφέρει στους χρήστες τη δυνατότητα μεταφοράς (migration) του ιστορικού τους μεταξύ διαφορετικών συσκευών με απόλυτα τοπικό τρόπο. Με αυτόν τον τρόπο, διασφαλίζεται η πλήρης φορητότητα των δεδομένων χωρίς να απαιτείται η δημιουργία cloud λογαριασμού, παραμένοντας σε απόλυτη στοίχιση με τη φιλοσοφία της εφαρμογής για μέγιστη προστασία της ιδιωτικότητας.

Πίνακας ορολογίας

Ξενόγλωσσος όρος	Ελληνικός Όρος	Έννοια / Επεξήγηση
Clean Architecture	Καθαρή Αρχιτεκτονική	Πρότυπο διαχωρισμού του κώδικα σε ανεξάρτητα επίπεδα (Data, Domain, Presentation).
Jetpack Compose	Jetpack Compose	Σύγχρονο, δηλωτικό (declarative) toolkit της Google για τη σχεδίαση native UI σε Android.
Room Database	Βάση Δεδομένων Room	Βιβλιοθήκη αφαίρεσης για την ασφαλή και εύκολη διαχείριση της τοπικής βάσης SQLite.
CameraX	Βιβλιοθήκη CameraX	Jetpack βιβλιοθήκη της Google που απλοποιεί την ενσωμάτωση λειτουργιών κάμερας.
Fuzzy Matching	Ασαφής Ταυτοποίηση	Αλγοριθμική τεχνική εύρεσης συμβολοσειρών που ταιριάζουν κατά προσέγγιση.
Levenshtein Distance	Απόσταση Levenshtein	Μετρική που υπολογίζει τον ελάχιστο αριθμό ενεργειών για τη μετατροπή μιας λέξης σε άλλη.
Kotlin Coroutines	Κορουτίνες Kotlin	Μηχανισμός της Kotlin για εκτέλεση ασύγχρονου κώδικα χωρίς μπλοκάρισμα του κύριου νήματος.
State	Κατάσταση	Οποιαδήποτε μεταβαλλόμενη τιμή δεδομένου που επηρεάζει άμεσα την εμφάνιση του UI.
Recomposition	Ανασύνθεση	Η επανεκτέλεση των composable συναρτήσεων για την ενημέρωση του UI λόγω αλλαγής του state.
Heuristics	Ευρετικοί Αλγόριθμοι	Πρακτικές μέθοδοι για την ταχεία επίλυση σύνθετων προβλημάτων με βάση κανόνες.
Android Studio	Android Studio	Το επίσημο ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) της Google για εφαρμογές Android.
Retrofit	Βιβλιοθήκη Retrofit	Βιβλιοθήκη τύπου HTTP client που απλοποιεί την ανταλλαγή δεδομένων με REST APIs.
Dagger Hilt	Framework Dagger Hilt	Framework έγχυσης εξαρτήσεων (Dependency Injection) ειδικά σχεδιασμένο για το Android.
DataStore (Preferences)	Αποθήκη Δεδομένων DataStore	Σύγχρονη λύση ασύγχρονης αποθήκευσης key-value δεδομένων που αντικαθιστά τα SharedPreferences.
Regular Expressions (Regex)	Κανονικές Εκφράσεις	Πρότυπα αναζήτησης χαρακτήρων για τον εντοπισμό και φιλτράρισμα λεκτικών μοτίβων σε κείμενο.
Base64	Κωδικοποίηση Base64	Σύστημα αναπαράστασης δυαδικών δεδομένων σε μορφή κειμένου ASCII για ασφαλή μεταφορά μέσω δικτύου.
Design Tokens	Σχεδιαστικά Διακριτικά	Οπτικές μεταβλητές (χρώματα, γραμματοσειρές) που ορίζουν ένα σχεδιαστικό σύστημα σε επίπεδο κώδικα.
Homoglyph	Ομόγλυφος Χαρακτήρας	Χαρακτήρας που μοιάζει οπτικά με άλλον, αλλά έχει διαφορετική κωδικοποίηση, προκαλώντας σφάλματα στο OCR.
N-Grams	Ν-Γράμματα	Ακολουθία Ν συνεχόμενων στοιχείων (λέξεων) από ένα κείμενο που χρησιμοποιείται για την εύρεση επωνυμιών καταστημάτων.

Πίνακας Συντμήσεων-Αρτικόλεξων-Ακρονυμίων

Σύντμηση / Ακρωνύμιο	Πλήρης Όρος (Αγγλικά)	Απόδοση στα Ελληνικά
API	Application Programming Interface	Διεπαφή Προγραμματισμού Εφαρμογών
CRUD	Create, Read, Update, Delete	Λειτουργίες Δημιουργίας, Ανάγνωσης, Ενημέρωσης, Διαγραφής
DAO	Data Access Object	Αντικείμενο Πρόσβασης Δεδομένων
DI	Dependency Injection	Έγχυση Εξαρτήσεων
JSON	JavaScript Object Notation	Μορφότυπος Ανταλλαγής Δεδομένων
MVVM	Model-View-ViewModel	Αρχιτεκτονικό Πρότυπο Σχεδίασης
OCR	Optical Character Recognition	Οπτική Αναγνώριση Χαρακτήρων
ORM	Object-Relational Mapping	Αντικειμενοστρεφής Χαρτογράφηση Σχέσεων
SDK	Software Development Kit	Πακέτο Ανάπτυξης Λογισμικού
UI	User Interface	Διεπαφή Χρήστη
URI	Uniform Resource Identifier	Ενιαίο Αναγνωριστικό Πόρων
UX	User Experience	Εμπειρία Χρήστη
XML	Extensible Markup Language	Επεκτάσιμη Γλώσσα Σήμανσης
CSV	Comma-Separated Values	Τιμές Διαχωρισμένες με Κόμμα
EXIF	Exchangeable Image File Format	Μορφότυπος Ανταλλάξιμων Αρχείων Εικόνας
HTTP	Hypertext Transfer Protocol	Πρωτόκολλο Μεταφοράς Υπερκειμένου
JPEG	Joint Photographic Experts Group	Κοινή Ομάδα Εμπειρογνωμόνων Φωτογραφίας
JVM	Java Virtual Machine	Εικονική Μηχανή Java
REST	Representational State Transfer	Αναπαραστατική Μεταφορά Κατάστασης
SQL	Structured Query Language	Δομημένη Γλώσσα Ερωτημάτων
ΦΠΑ	Value Added Tax (VAT)	Φόρος Προστιθέμενης Αξίας

Βιβλιογραφία

Alepis, E., & Nita, S. (2017). Mobile application providing accessible routes for people with mobility impairments. *2017 8th International Conference on Information, Intelligence, Systems & Applications (IISA)*. doi:https://doi.org/10.1109/IISA.2017.8316439

Alepis, E., & Virvou, M. (2012). Multimodal Object Oriented User Interfaces in Mobile Affective Interaction. *Multimedia Tools and Applications*. doi:https://doi.org/10.1007/s11042-011-0744-y

Android Developers. (2025). *CameraX overview*. Ανάκτηση από https://developer.android.com/media/camera/camerax

Android Developers. (2025). *Dependency injection with Hilt*. Ανάκτηση από https://developer.android.com/training/dependency-injection/hilt-android

Android Developers. (2025). *Guide to app architecture*. Ανάκτηση από https://developer.android.com/topic/architecture

Android Developers. (2025). *Jetpack Compose*. Ανάκτηση από https://developer.android.com/compose

Android Developers. (2025). *Room persistence library*. Ανάκτηση από https://developer.android.com/training/data-storage/room

Dagger. (χ.χ.). *Hilt Application Quick Start*. Ανάκτηση από https://dagger.dev/hilt/quick-start.html

Elizarov, R. A. (2024). *Kotlin in Action* (2 εκδ.). Shelter Island, NY, U.S.A.: Manning Publications.

GeeksforGeeks. (2024). *Complete Guide to Clean Architecture.* Ανάκτηση από https://www.geeksforgeeks.org/system-design/complete-guide-to-clean-architecture/

GeeksforGeeks. (2024). *MVVM (Model View ViewModel) Architecture Pattern in Android*. Ανάκτηση από https://www.geeksforgeeks.org/android/mvvm-model-view-viewmodel-architecture-pattern-in-android/

Google Cloud. (2025). *Cloud Vision documentation*. Ανάκτηση από https://cloud.google.com/vision/docs

Google Cloud. (2025). *Optical Character Recognition (OCR).* Ανάκτηση από https://cloud.google.com/vision/docs/ocr

Google Developers. (2024). *State and Jetpack Compose*. Ανάκτηση από https://developer.android.com/develop/ui/compose/state

Google Developers. (2025). *App Architecture: Data Layer - DataStore*. Ανάκτηση από https://developer.android.com/topic/libraries/architecture/datastore

JetBrains. (2025). *Kotlin documentation*. Ανάκτηση από https://kotlinlang.org/docs/home.html

JUnit Team. (2025). *JUnit 4*. Ανάκτηση από https://junit.org/junit4/](https://junit.org/junit4/

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics Doklady. *Soviet Physics Doklady, 10*(8), 707–710.

Martin, R. C. (2018). *Clean Architecture: A Craftsman's Guide to Software Structure and Design.* Boston, MA, U.S.A.: Prentice Hall.

Sills, B. G. (2023). *Android Programming: The Big Nerd Ranch Guide* (5 εκδ.). Atlanta, GA, U.S.A.: Big Nerd Ranch, LLC.

Square Inc. (2025). *Retrofit Documentation*. Ανάκτηση από https://square.github.io/retrofit/declarations/