ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Δημιουργία εφαρμογής σε Android Studio η οποία υπολογίζει τα κοινόχρηστα μιας πολυκατοικίας.

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

των

ΑΘΑΝΑΣΙΟΣ ΝΟΥΣΙΟΣ 2056

&

ΠΟΛΥΧΡΟΝΗΣ ΕΥΚΑΡΠΙΔΗΣ 2025

Επιβλέπων : ΔΗΜΗΤΡΙΟΣ Ι. ΒΕΡΓΑΔΟΣ

ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΗΣ

Καστοριά Μάιος - 2024

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Δημιουργία εφαρμογής σε Android Studio η οποία υπολογίζει τα κοινόχρηστα μιας πολυκατοικίας.

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

των

ΑΘΑΝΑΣΙΟΣ ΝΟΥΣΙΟΣ 2056

&

ΠΟΛΥΧΡΟΝΗΣ ΕΥΚΑΡΠΙΔΗΣ 2025

Επιβλέπων : ΔΗΜΗΤΡΙΟΣ Ι. ΒΕΡΓΑΔΟΣ
ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΗΣ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 20 Μαΐου 2024

Δημήτριος Ι. Βέργαδος	Ιωάννης Βαρδάκας	Νίκος Δημόκας
Επίκουρος Καθηγητής	Αναπληρωτής Καθηγητής	Επίκουρος Καθηγητής

Καστοριά Μάιος - 2024

Ευχαριστίες

Θα θέλαμε να ευχαριστήσουμε τον κύριο Δημήτριο Βέργαδο επιβλέποντα της πτυχιακής μας εργασίας, για τη στήριξη και την καθοδήγηση του καθ’ όλη τη διάρκεια εκπόνησης της εργασίας.

Περίληψη

Στην παρούσα πτυχιακή εργασία έγινε σχεδίαση, ανάλυση, υλοποίηση και αξιολόγηση μιας εφαρμογής η οποία είναι προσβάσιμη από συσκευές με λογισμικό Android. Η εφαρμογή παρέχει τη δυνατότητα δημιουργίας λογαριασμού από το χρήστη με την εισαγωγή ενός μόνο username και στη συνέχεια, του επιτρέπει την καταχώρηση κτιρίων ή πολυκατοικιών που διαχειρίζεται. Δίνοντας του επίσης την επιλογή να ορίσει ο ίδιος τα ποσοστά χρέωσης για τον κάθε ένοικο, ή να υπολογίζει το κόστος βάση νόμου χρησιμοποιώντας σαν κριτήριο τον όροφο και τα τετραγωνικά. Αφού γίνει η πρώτη καταχώρηση, ο χρήστης μπορεί να προχωρήσει στην εισαγωγή των ενοίκων, με τις απαραίτητες πληροφορίες για τον καθένα, όπως ονοματεπώνυμο, στοιχεία επικοινωνίας, όροφο, τετραγωνικά μέτρα κ.α. Στη συνέχεια, ο χρήστης καλείται να συμπληρώσει τα απαραίτητα πεδία για τον υπολογισμό του συνολικού κόστους, τα οποία διαμορφώνονται αναλόγως με τις επιλογές του. Με τη συμπλήρωση αυτών, η εφαρμογή υπολογίζει το κόστος και το παρουσιάζει σε έναν πίνακα όπου κάθε γραμμή αντιστοιχεί σε έναν ένοικο, μοιράζοντας το συνολικό κόστος στον καθένα και αναλύοντας πως προκύπτει το κάθε ποσό. Τέλος ο χρήστης μπορεί να κατεβάσει τον πίνακα στη συσκευή του σε μορφή PDF και να τον μοιραστεί με τους υπόλοιπους ενοίκους, είτε μέσω αυτής είτε σε κάποιο πίνακα ανακοινώσεων της οικοδομής.

Αρχική ιδέα της εφαρμογής ήταν η σύγχυση που επικρατεί στο χώρο αυτό, κυρίως στα αστικά κέντρα της χώρας τα οποία κατακλύζονται από πολυκατοικίες, καθώς επίσης και η καινοτομία που φέρνει μαζί της, διότι πέρα από εταιρείες διαχείρισης οι οποίες έρχονται με ένα κόστος, δεν υπάρχει κάποια άλλη εναλλακτική. Σκοπός λοιπόν της εφαρμογής είναι η διευκόλυνση του υπολογισμού των κοινοχρήστων μιας πολυκατοικίας χωρίς κάποιο κόστος, κάνοντας τον ένοικο και διαχειριστή. Σε επόμενο κεφάλαιο έγινε ανάλυση και σχεδίαση της εφαρμογής με διάφορες τεχνικές, όπως με διαγράμματα UML και στη συνέχεια ακολούθησε η

υλοποίηση της χρησιμοποιώντας τις τεχνολογίες και τις γλώσσες προγραμματισμού που είχαν διερευνηθεί. Τέλος, έγινε αξιολόγηση της εφαρμογής από χρήστες με τη χρήση του APK, αλλά και ανάλυση της αξιολόγησης.

Για την υλοποίηση της εφαρμογής χρησιμοποιήθηκαν διάφορα τεχνολογικά εργαλεία, όπως το Android Studio το οποίο αποτελεί τα θεμέλια για τη σύστασή της, η γλώσσα σήμανσης XML για τη δημιουργία του UI (User Interface), η γλώσσα προγραμματισμού Kotlin για να γίνει λειτουργικό το UI καθώς και για τη σύνδεση με τη βάση δεδομένων και τέλος η βάση δεδομένων που χρησιμοποιήθηκε είναι το σχεσιακό σύστημα διαχείρισης βάσεων δεδομένων SQLite.

Λέξεις Κλειδιά:

Εφαρμογή Android, Ανάπτυξη Εφαρμογών Android, Προγραμματισμός για Φορητές Συσκευές, Υπολογισμός Κοινοχρήστων, Kotlin, Android Studio IDE, XML, SQLite.

Abstract

In this thesis, design, analysis, implementation, and evaluation of an application accessible from Android devices were conducted. The application allows users to create an account by entering only one username and then enables them to register buildings or apartment buildings they manage. It also gives them the option to define charging percentages for each tenant themselves or to calculate costs based on law criteria such as floor and square meters. Once the initial registration is done, the user can proceed to enter tenants, with necessary information for each, such as name, contact details, floor, square meters, etc. Then, the user is prompted to fill in the necessary fields for calculating the total cost, which varies depending on the user's choices. With this information filled in, the application calculates the cost and presents it in a table where each row corresponds to a tenant, distributing the total cost to each one and analyzing how each amount is derived. Finally, the user can download the table in PDF format to their device and share it with other tenants, either through the application or on a notice board in the building.

The initial idea of the application stemmed from the confusion prevailing in this area, especially in urban centers flooded with apartment buildings, and the innovation it brings, as apart from property management companies that come with a cost, there is no other alternative. Therefore, the aim of the application is to facilitate the calculation of common expenses for an apartment building without any cost, involving both the tenant and the manager. In the next chapter, the application was analyzed and designed using various techniques, such as UML diagrams, followed by its implementation using the explored technologies and programming languages. Finally, the application was evaluated by users using the APK, and the evaluation analysis was conducted.

Various technological tools were used for the implementation of the application, such as Android Studio, which forms the foundation for its

establishment, XML markup language for creating the UI (User Interface), Kotlin programming language to make the UI functional and for connecting to the database, and finally, the database used is the SQLite relational database management system.

Keywords:

Android Application, Android App Development, Mobile Device Programming, Common Expenses Calculation, Kotlin, Android Studio IDE, XML, SQLite.

Πίνακας Περιεχομένων

Λίστα Εικόνων

Συντομογραφίες & Ακρωνύμια

JVM	Java Virtual Machine
IOS	iPhone Operating System
SQL	Structured Query Language
MFC	Maintenance Fees Calculator
PDF	Portable Document Format
UI	User Interface
ERD	Entity Relationship Diagrams
XML	Extensible Markup Language
HTML	Hyper Text Markup Language

Εισαγωγή

Οι πολυκατοικίες όπως πλέον τις γνωρίζουμε άρχισαν τα κάνουν την εμφάνιση τους στην Ελλάδα στις αρχές του 20^{ου} αιώνα, ανήκοντας σε έναν μόνο ιδιοκτήτη. Με την πάροδο των χρόνων και κυρίως μετά το νόμο για την οριζόντια ιδιοκτησία που θέσπισε το κράτος, απελευθέρωσε τις εργολαβίες και ώθησε στην κατασκευή ολοένα και περισσότερων πολυκατοικιών[34]. Πλησιάζοντας προς τα τέλη του 1980 τα αστικά κέντρα της Ελλάδας άλλαξαν όχι μόνο στην όψη αλλά και στην κοινωνική γεωγραφία τους. Οι αλλαγές αυτές έφεραν την μετάβαση των περισσότερων νοικοκυριών από τη μονοκατοικία στο διαμέρισμα και κατ' επέκταση στη δημιουργία του όρου κοινόχρηστα. Ανεξάρτητα λοιπόν από το αν στο διαμέρισμα διαμένει ο ιδιοκτήτης του ή κάποιος ένοικος, η πολυκατοικία έχει έξοδα συντήρησης τα οποία πρέπει να καλύπτονται για την ομαλή λειτουργία της.

Έτσι δημιουργήθηκε η ανάγκη για κάποιον διαχειριστή, ο οποίος θα ήταν υπεύθυνος για την οργάνωση και την ενημέρωση όλων όσων διαμένουν στη πολυκατοικία και έτσι, το ρόλο αυτό ήρθαν να παίξουν εταιρείες διαχείρισης κοινοχρήστων. Πλέον κάθε χρόνο όλο και περισσότερες πολυκατοικίες συνεργάζονται με εταιρείες διαχείρισης κοινοχρήστων που τους κρατάνε προμήθεια, με αποτέλεσμα να αυξάνονται τα έξοδα του νοικοκυριού κάθε μήνα. Γιατί να μην μπορούσε ένας ένοικος της πολυκατοικίας να κάνει τον διαχειριστή, γρήγορα και απλά;

Η τεχνολογία γίνεται ολοένα και μεγαλύτερο κομμάτι της καθημερινότητας των ανθρώπων, με τη χρήση έξυπνων κινητών τηλεφώνων να διεκδικεί μεγάλο μέρος από αυτή. Με τη χρήση τους πολλές υπηρεσίες έχουν διευκολυνθεί και η καθημερινότητα του ανθρώπου έχει γίνει πιο εύκολη. Αυτό οφείλεται στο ότι μια τόσο μικρή συσκευή συνδυάζει τόσες πολλές παροχές και κάνει το έξυπνο κινητό τηλέφωνο χρηστικό σε όλες τις πτυχές της ζωής του ανθρώπου. Η διαχείριση των κοινοχρήστων δεν είναι σε καμία περίπτωση εξαίρεση σε αυτό.

Σε σχέση με τον παραδοσιακό υπολογισμό των κοινοχρήστων, ο υπολογισμός μέσο μιας εφαρμογής διαθέτει αρκετά προνόμια. Τα σημαντικότερο από αυτά είναι η αποδοτικότητα και η έλλειψη της ανάγκης να κρατάς αρχείο σε απλή μορφή με αποτέλεσμα να γεμίζεις με χαρτιά. Άλλο ένα πολύ σημαντικό πλεονέκτημα είναι ότι μπορείς να το έχεις πάντα μαζί σου και να ανατρέξεις σε αυτό ανά πάσα στιγμή που το θελήσεις, απλά και μόνο με τη χρήση του κινητού σου τηλεφώνου. Φυσικά όλα αυτά τα προνόμια γίνονται αντιληπτά όταν αξιοποιούνται οι τελευταίες τεχνολογίες στη συσκευή, τόσο σε λογισμικό όσο και υλικό επίπεδο.

Η εφαρμογή που παρουσιάζουμε δίνει την δυνατότητα σε κάποιο μέλος – ένοικο της πολυκατοικίας να μπορεί να γίνει διαχειριστής χωρίς κάποιο κόπο και επιπλέον έξοδα. Ο χρήστης να μπορεί καταχωρίσει και να επεξεργαστεί εύκολα και απλά τα στοιχεία της πολυκατοικίας και των ενοίκων. Να ανατρέξει ανά πάσα στιγμή σε παρελθοντικά αρχεία και να τα διαμοιράσει στους ενοίκους σε ηλεκτρονική μορφή.

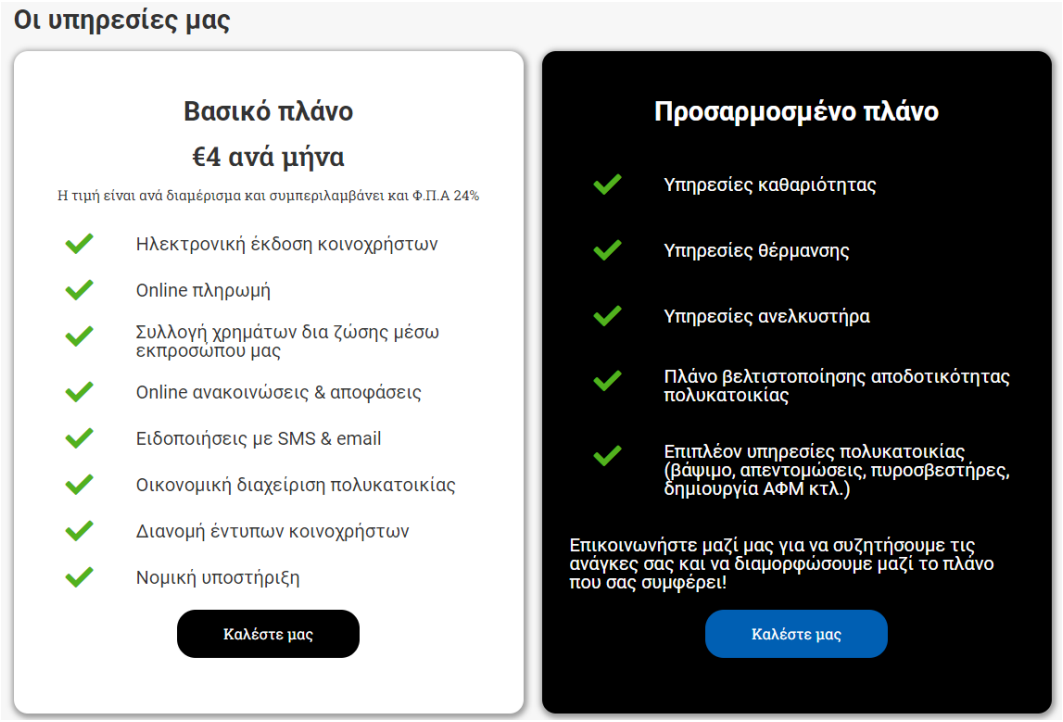
Παρόμοιες Εφαρμογές

Όπως αναφέρθηκε και παραπάνω στην εισαγωγή, η εναλλακτική επιλογή για τη διαχείριση των κοινοχρήστων είναι οι ανάλογες εταιρείες στον σχετικό τομέα. Οι εταιρίες αυτές, εκτός από υπολογισμό και έκδοση των κοινοχρήστων παρέχουν και υπηρεσίες όπως συνεργεία καθαρισμού, τεχνική συντήρηση, νομική υποστήριξη κ.α. για τις πολυκατοικίες καθώς επίσης και υπηρεσίες για κάποιο διαμέρισμα μεμονωμένο. Αυτές οι εταιρίες έχουν σαν πολιτική λειτουργίας να χρεώνουν το κάθε διαμέρισμα ξεχωριστά, ένα έξτρα ποσό ανάλογα με τις υπηρεσίες που θα διαλέξει η εκάστοτε πολυκατοικία.

Παρακάτω ακολουθούν σχετικά παραδείγματα με εταιρείες διαχείρισης κοινοχρήστων.

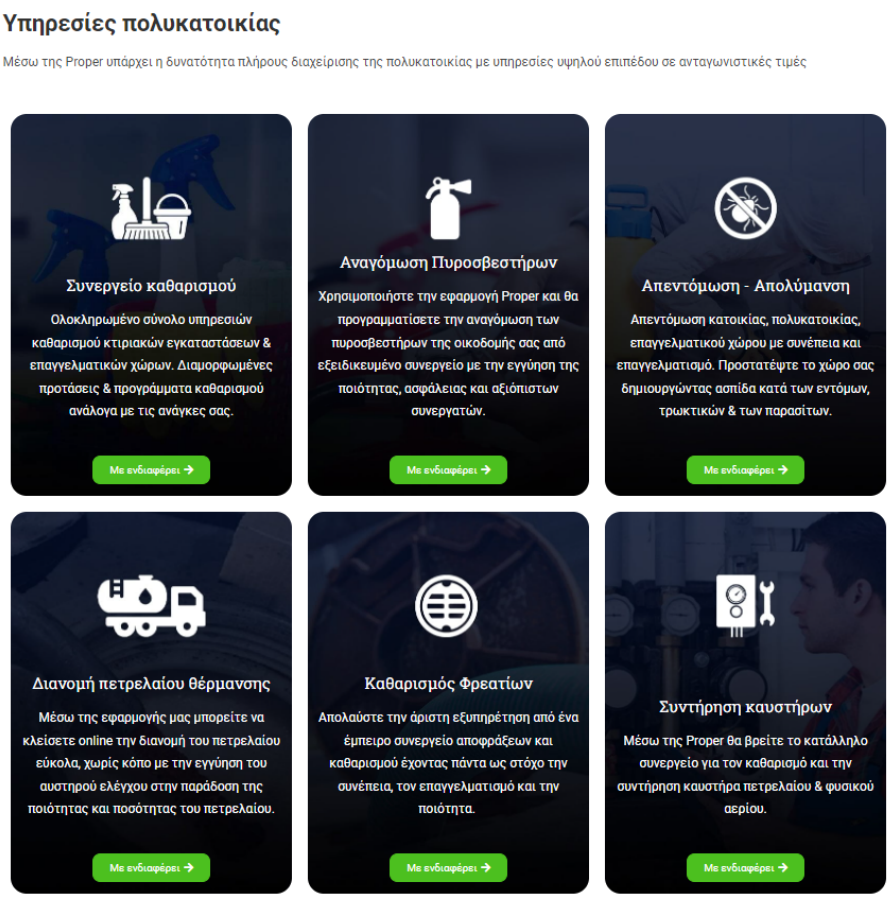
Παράδειγμα 1: Proper (https://proper.gr)

Εδώ φαίνονται τα δύο πλάνα που μπορεί να προσφέρει η εταιρία Proper[36]. Το βασικό πλάνο που παρέχει τις βασικές υπηρεσίες που χρειάζεται μια πολυκατοικία και το προσαρμοσμένο που μπορεί ο χρήστης – πολυκατοικία να προσαρμόσει ανάλογα με τις ανάγκες και τις οικονομικές δυνατότητες της.



Εικόνα 1 Πλάνα χρέωσης Proper

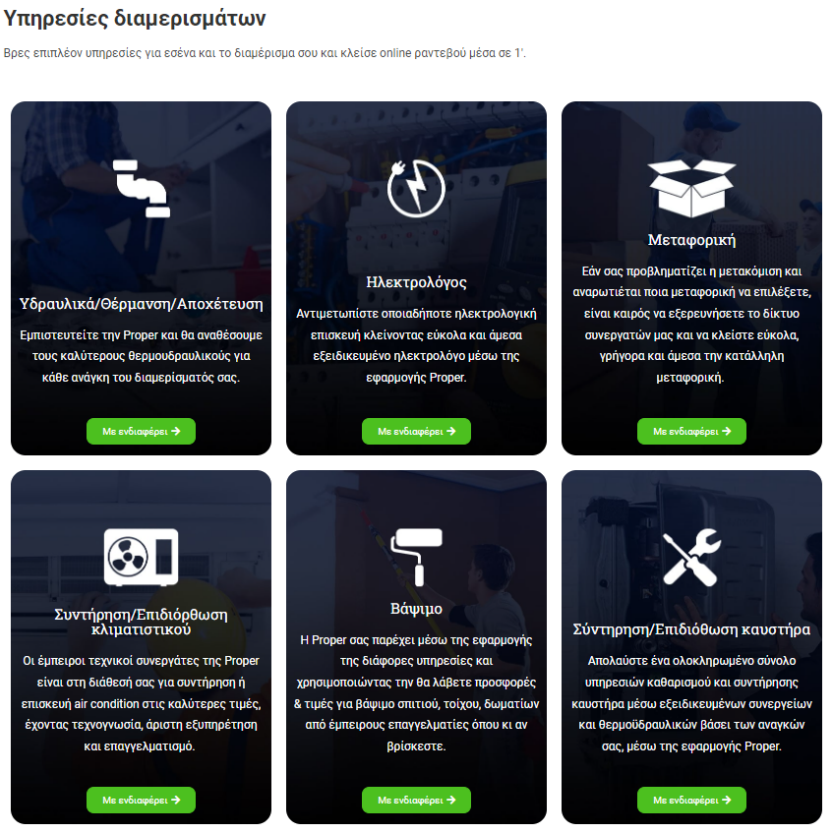
Πιο αναλυτικά, οι υπηρεσίες που μπορεί να παρέχει η Proper σε μία πολυκατοικία, βλέποντας και παρακάτω την σχετική εικόνα, είναι το συνεργείο καθαρισμού, αλλαγή πυροσβεστήρων που είναι υποχρεωτικοί από το νόμο, απεντόμωση, διανομή πετρελαίου, καθαρισμός φρεατίων και συντήρηση καυστήρα.



Εικόνα 2 Υπηρεσίες Πολυκατοικίας Proper

Με μία επιπλέον έξτρα χρέωση υπάρχει η δυνατότητα και για παροχή άλλων υπηρεσιών που αφορούν συγκεκριμένα μόνο ένα διαμέρισμα. Με αυτό το τρόπο μπορούν να αποφευχθούν τυχόν συνελεύσεις των ενοίκων, καθώς και η πληρωμή επιπλέον δαπανών για διαμερίσματα τρίτων. Οι υπηρεσίες αυτές διατίθενται για το κάθε διαμέρισμα ξεχωριστά, με τη δική του μόνο θέληση αλλά και χρέωση. Βλέποντας την παρακάτω εικόνα μπορούμε να διακρίνουμε τις υπηρεσίες, όπως τεχνική υποστήριξη για βλάβες σχετικά με υδραυλικά, θέρμανση και αποχέτευση. Δηλαδή, η Proper αναλαμβάνει την διεκπεραίωση των εργασιών αυτών, σε συνεργασία με τον εκάστοτε επαγγελματία, π.χ. ηλεκτρολόγο, για όλες τις ανάγκες του ενοίκου. Ακόμη, η εταιρεία παρέχει υπηρεσίες μεταφοράς πραγμάτων από και

προς το διαμέρισμά, όπως επίσης και υπηρεσίες βαψίματος, αν και εφόσον το επιθυμεί ο ένοικος.

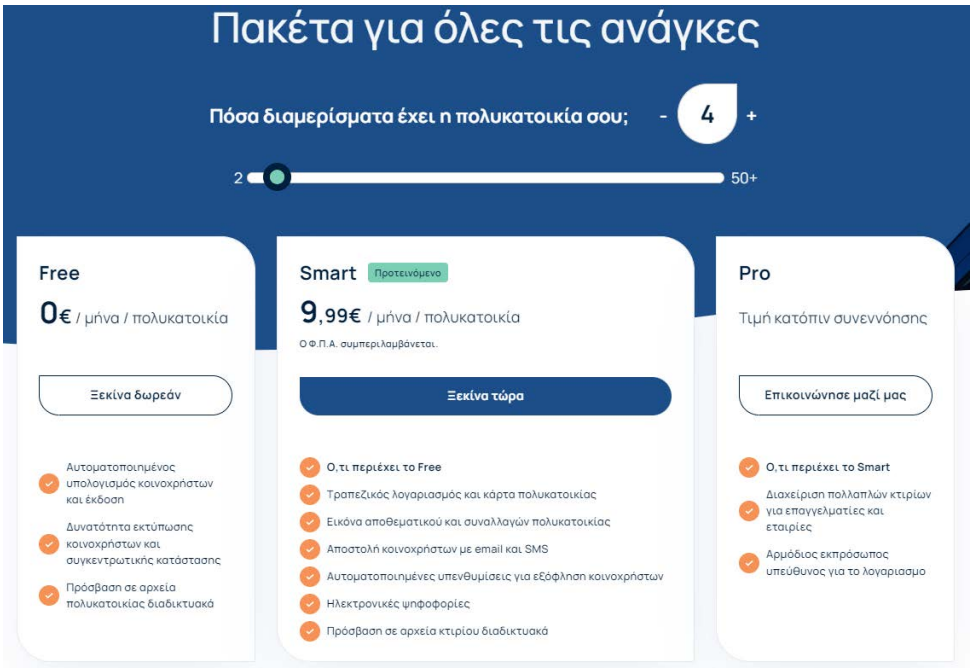


Εικόνα 3 Υπηρεσίες Διαμερισμάτων Proper

Παράδειγμα 2: Billys (https://billys.gr)

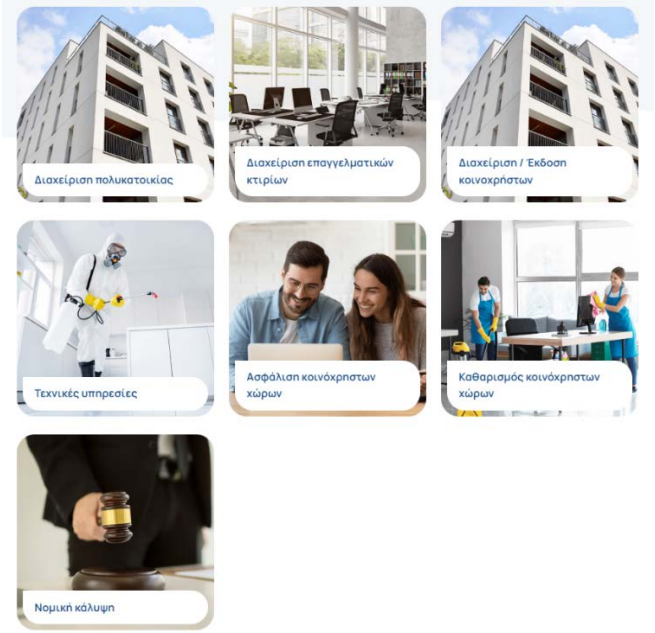
Η Billys[37] είναι μια άλλη εταιρία παροχής υπηρεσιών κοινοχρήστων για πολυκατοικίες, αλλά σε αυτή τη περίπτωση έχουμε μερικές διαφοροποιήσεις. Εδώ έχουμε τρία διαφορετικά πλάνα. Το πρώτο πλάνο που διατίθεται και δωρεάν, παρέχει υπολογισμό και έκδοση των κοινοχρήστων, εκτύπωση της κατάστασης και πρόσβαση στα αρχεία της πολυκατοικίας ηλεκτρονικά. Με το πλάνο smart παρέχεται το πλάνο free μαζί με αποστολή των κοινοχρήστων με SMS και Email, υπενθύμιση

εξόφλησης του λογαριασμού και ηλεκτρονικές ψηφοφορίες σε περίπτωση που δεν μπορεί ο εκάστοτε ένοικος να παραβρεθεί στις συναντήσεις. Ακόμη, προσφέρει στον διαχειριστή έναν λογαριασμό με κάρτα σε τράπεζα, στην οποία μαζεύονται όλα τα έξοδα, με αποτέλεσμα να έχει μία «εικόνα» για το αποθεματικό και τις συναλλαγές της πολυκατοικίας. Το καλύτερο πλάνο που διαθέτει είναι το pro, το οποίο παρέχει ότι και το smart αλλά με την διαφορά ότι διαθέτει επιπρόσθετα διαχείριση για πολλαπλά κτίρια για επαγγελματίες ή εταιρίες, όπως κτήριο με γραφεία, και προσφέρει αρμόδιο εκπρόσωπο, υπεύθυνο για την διαχείριση του λογαριασμού και την συγκέντρωση των κοινοχρήστων.



Εικόνα 4 Υπηρεσίες Πολυκατοικίας Billys

Παρακάτω βλέπουμε ότι διαθέτει επιπλέον υπηρεσίες καθαρισμού, ασφάλισης κοινόχρηστων χώρων και επιπλέον νομική κάλυψη για τους ιδιοκτήτες.



Εικόνα 5 Διάφορες παροχές Billys

Τεχνολογίες Υλοποίησης της εφαρμογής

Android Studio

Το Android Studio[2] είναι το επίσημο ολοκληρωμένο προγραμματιστικό περιβάλλον (IDE) για την ανάπτυξη Android εφαρμογών της Google. Είναι διαθέσιμο σε λειτουργικά συστήματα Windows, macOS και Linux, ενώ ανακοινώθηκε η κυκλοφορία του από την Google στις 16 Μαΐου του 2013 σε συνέδριο της Google στο Σαν Φρανσίσκο της Καλιφόρνιας. Από τον Μάιο του 2013 ξεκίνησε με την έκδοση 0.1 και η beta έκδοση 0.8 κυκλοφόρησε τον Ιούνιο του 2014. Η πρώτη σταθερή της έκδοση 1.0 βγήκε το Δεκέμβριο του 2014. Επίσης, στις 7 Μαΐου του 2019 η Kotlin αντικατέστησε τη Java ως προτεινόμενη και κύρια γλώσσα του android studio, αξακολουθώντας όμως να υποστηρίζει την γλώσσα java όπως και την C++. Αυτή τη στιγμή βρίσκεται στην έκδοση Iguana|2023.2.1

Σχεδιάστηκε αποκλειστικά για προγραμματισμό Android και αντικατέστησε το Eclipse ως το κύριο ολοκληρωμένο προγραμματιστικό περιβάλλον για ανάπτυξη εφαρμογών Android. Έχει σαν βάση το λογισμικό της JetBrains Intellij της IDEA με αποτέλεσμα να κληρονομεί χαρακτηρίστηκα όπως:

- Ισχυρή επεξεργασία κώδικα (Powerful Code Editing) : Σύστημα αυτόματης συμπλήρωσης κώδικα, πλοήγηση στα αρχεία του έργου, προηγμένη και ασφαλή επεξεργασία και προεπισκόπηση πόρων.
- Ανάλυση κώδικα κατά την επεξεργασία (On-The-fly Code Analysis) : Εμφάνιση προειδοποιήσεων και σφαλμάτων στον κώδικα κατά την πληκτρολόγηση με επισήμανση προτεινόμενης αυτόματης επίλυσης.

- Ενσωματωμένα εργαλεία (Built-in Android Tools) : Ενσωματωμένο σύστημα καταγραφής Logcat, σχεδιαστή διεπαφής με drag-n-drop, εικονικές μηχανές (Virtual Devices)

Άλλα χαρακτηριστικά του Android Studio είναι το σύστημα Proguard για την συρρίκνωση και βελτιστοποίηση του κώδικα, η ψηφιακή υπογραφή της εφαρμογής, διάφορα πρότυπα κώδικα (Code Templates) για την ανάπτυξη εφαρμογών, η προβολή της βάσης δεδομένων του προγράμματος (App Inspection) και η διόρθωση της, ενώ τρέχει η εφαρμογή. Παρέχει εργαλεία αναδιαμόρφωσης τα οποία συμβάλλουν στις έξυπνες και γρήγορες επιδιορθώσεις του κώδικα, όπως το εργαλείο μετονομασίας μιας μεταβλητής που την αντικαθιστά σε ολόκληρο το κώδικα. Ακόμη, περιλαμβάνει εύκολο σχεδιασμό με drag & drop ή με κώδικα XML του UI της οθόνης και χρησιμοποιεί το Gradle που βοηθάει στην κατασκευή καλύτερου λογισμικού γρηγορότερα. Τέλος, δημιουργεί αρχεία APK τα οποία χρησιμοποιούνται από το λειτουργικό android για την εγκατάσταση της εφαρμογής.

Ένα από τα μεγαλύτερα υπέρ του android studio είναι η άμεση σύνδεση του με το GitHub, μια πλατφόρμα που χρησιμοποιείται από πολλούς προγραμματιστές. Το GitHub προσφέρει οργανωμένο και κατανεμημένο έλεγχο των εκδόσεων του κώδικα και αποθήκευση του σε cloud. Το android studio δίνει τη δυνατότητα με το πάτημα ενός κουμπιού να ανεβάσει ο χρήστης το πρόγραμμα του μαζί με τα σχόλια που επιθυμεί σχετικά με την έκδοση στην πλατφόρμα.

Εκδόσεις του Android Studio

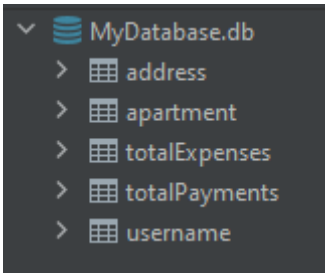
Όπως αναφέρθηκε και παραπάνω, η πρώτη επίσημη έκδοση του Android Studio 1.0[4] κυκλοφόρησε το Δεκέμβριο του 2014 και ήταν βασισμένη στην έκδοση 1.8 του Intellij IDEA. Με την έκδοση 2.0 που κυκλοφόρησε τον Απρίλιο του 2016

βελτιώθηκε το σε ότι αφορά την ταχύτητα, την απόδοση και το design του. Με επιπλέον αναβαθμίσεις πάνω σε αυτή την έκδοση το Σεπτέμβριο του 2016 προστέθηκαν νέα χαρακτηριστικά όπως το Layout Editor, το Constraint Layout και άλλες βελτιώσεις για τη σταθερότητα του. Η έκδοση 3.0 κυκλοφόρησε το Οκτώβριο του 2017 και μέχρι τον Αύγουστο του 2019 και την έκδοση 3.5 έφερε αναβαθμίσεις και προσθέσεις όπως το νέο Android Profiler, το Android App Bundle και το Energy Profiler. Με την έκδοση 4.0 τον Μάιο του 2020, μέχρι και τον Οκτώβριο του 2021 προστέθηκαν το Motion Editor, το Build Analyzer, το Jetpack Compose Preview και το SQL Database Inspector. Μερικές από τις κύριες ενσωματωμένες τεχνολογίες και χαρακτηριστικά που εισήχθησαν κατά τη διάρκεια των εκδόσεων 2020.3 - Arctic Fox από τον Ιούλιο 2021, περιλαμβάνουν την εισαγωγή του jetpack compose ως επίσημο λειτουργικό σύστημα για τη δημιουργία UI σε εφαρμογές Android. Από τότε μέχρι και σήμερα υπάρχουν συνεχείς ενημερώσεις επιτρέποντας στους προγραμματιστές να αναπτύσσουν εφαρμογές Android με πιο αποτελεσματικό και καινοτόμο τρόπο. Για την υλοποίηση της παρούσας εφαρμογής χρησιμοποιήθηκε η έκδοση Iguana | 2023.2.1 Patch 1[1]

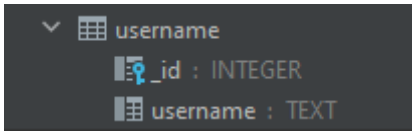
Εργαλεία του Android Studio που Χρησιμοποιήθηκαν

App Inspection

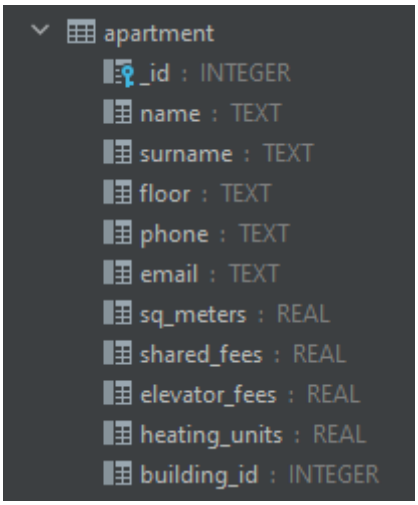
Το app inspection[3] είναι ένα εργαλείο του android studio που μας δίνει πρόσβαση στη βάση δεδομένων και στις εργασίες του παρασκηνίου. Μας επιτρέπει να επιβλέπουμε, να ρωτήσουμε και να τροποποιήσουμε τη βάση δεδομένων της εφαρμογής μας κατά την εκτέλεση της.



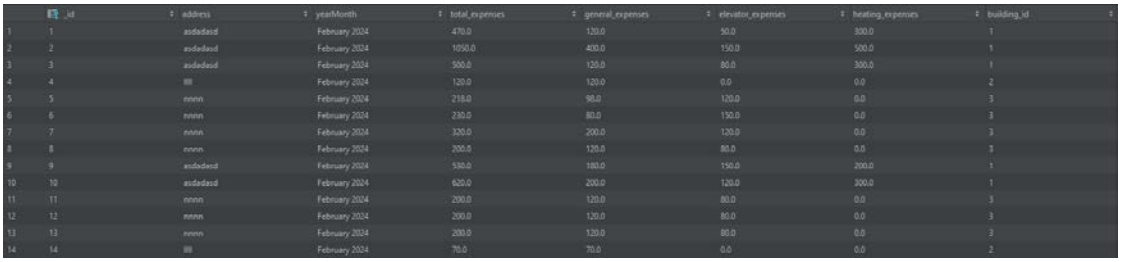
Εικόνα 6 Απεικόνιση App Inspection Data πίνακες



Εικόνα 7 Απεικόνιση App Inspection Data Username



Εικόνα 8 Απεικόνιση App Inspection Data apartment

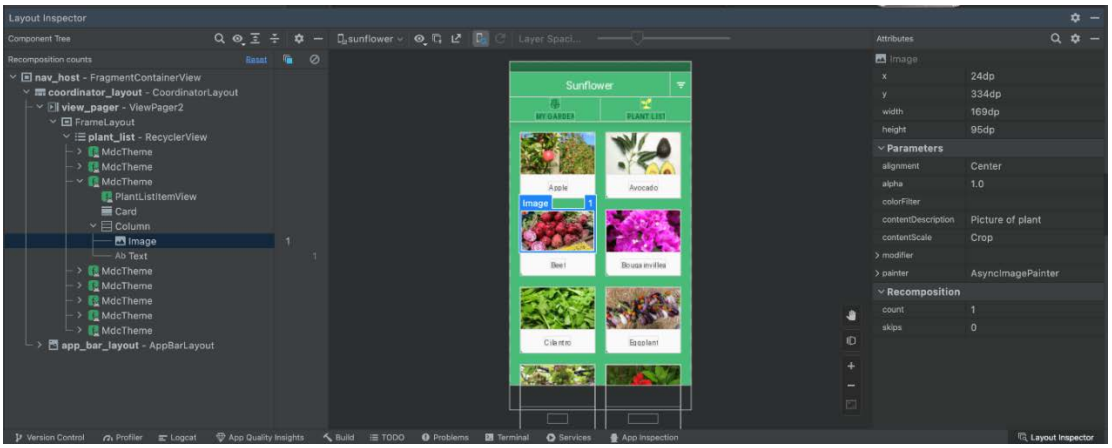


	_id	address	yearMonth	total_expenses	general_expenses	elevator_expenses	heating_expenses	building_id
1	1	asdadasd	February 2024	470.0	120.0	50.0	300.0	1
2	2	asdadasd	February 2024	1050.0	400.0	150.0	500.0	1
3	3	asdadasd	February 2024	500.0	120.0	80.0	300.0	1
4	4	iii	February 2024	120.0	120.0	0.0	0.0	2
5	5	nnnn	February 2024	218.0	98.0	120.0	0.0	3
6	6	nnnn	February 2024	230.0	80.0	150.0	0.0	3
7	7	nnnn	February 2024	320.0	200.0	120.0	0.0	3
8	8	nnnn	February 2024	200.0	120.0	80.0	0.0	3
9	9	asdadasd	February 2024	530.0	180.0	150.0	200.0	1
10	10	asdadasd	February 2024	620.0	200.0	120.0	300.0	1
11	11	nnnn	February 2024	200.0	120.0	80.0	0.0	3
12	12	nnnn	February 2024	200.0	120.0	80.0	0.0	3
13	13	nnnn	February 2024	200.0	120.0	80.0	0.0	3
14	14	iii	February 2024	70.0	70.0	0.0	0.0	2

Εικόνα 9 Απεικόνιση App Inspection Data total expenses

Layout Inspector

Το Layout Inspector[5] είναι ένα εργαλείο που παρέχει μια οπτική 3D απεικόνιση της ιεραρχίας των συστατικών που απαρτίζουν τη διάταξη της διεπαφής χρήστη.



Εικόνα 10 Layout Inspector

Refactor

Η λειτουργία "Refactor"[22] στο Android Studio παρέχει ένα σύνολο εργαλείων που επιτρέπουν την αναδιάταξη και την αναδιοργάνωση του κώδικα με ασφάλεια και ακρίβεια. Μέσω των βασικών λειτουργιών "Refactor", όπως το Rename, το Extract, το Move, το Copy, το Inline και το Introduce Variable, οι προγραμματιστές μπορούν να επεξεργαστούν και να αλλάξουν τη δομή του κώδικα χωρίς να χρειάζεται να τον αλλάξουν χειροκίνητα σε πολλά σημεία του προγράμματος. Αυτό ενισχύει την ποιότητα του κώδικα, την αποτελεσματικότητα και τη συντήρηση του.

Logcat

Το εργαλείο Logcat[6] μπορεί να ρυθμιστεί ώστε να εμφανίζει όλα τα γεγονότα που σχετίζονται με το τρέξιμο της εφαρμογής, τη συσκευή ή τον εξομοιωτή ή να περιοριστεί σε εκείνα που αφορούν την τρέχουσα επιλεγμένη εφαρμογή. Η έξοδος μπορεί επίσης να περιοριστεί μόνο στα συμβάντα καταγραφής που ταιριάζουν με ένα καθορισμένο φίλτρο. Όπως βλέπουμε και στην παρακάτω εικόνα κώδικας που έχουμε προσθέσει στις μεθόδους των activity είναι για να εξάγει πληροφορίες καταγραφής στο παράθυρο του εργαλείου logcat.

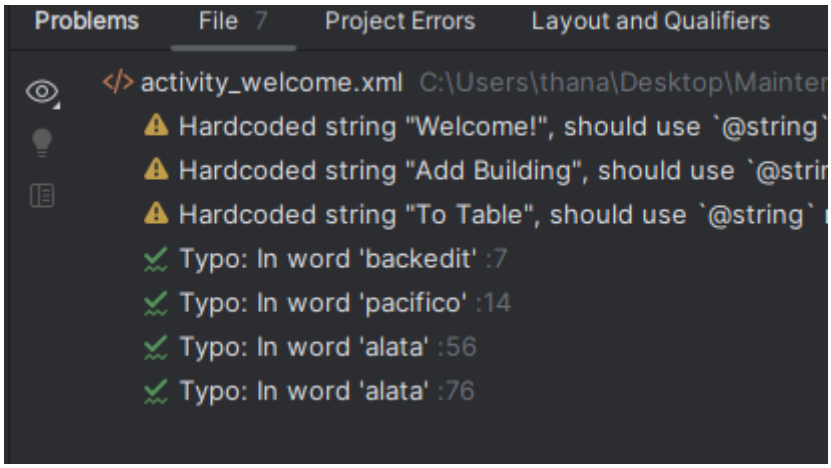
Εικόνα 11 Εργαλείο Logcat

```
111             val intent = Intent( packageContext: this, CalculateActivity::class.java)
112             intent.putExtra( name: "ELEVATOR_CHECKED", elevatorChecked)
113             intent.putExtra( name: "HEATING_CHECKED", heatingChecked)
114             intent.putExtra( name: "M2CALC_CHECKED", m2calcChecked)
115             intent.putExtra( name: "BUILDING_ID", buildingId)
116
117             Log.d( tag: "ApartmentsActivity", msg: "Starting CalculateActivity with intent values:" +
118                     "\nELEVATOR_CHECKED: $elevatorChecked" +
119                     "\nHEATING_CHECKED: $heatingChecked" +
120                     "\nM2CALC_CHECKED: $m2calcChecked" +
121                     "\nBUILDING_ID: $buildingId")
122             startActivity(intent)
123         }
```

Εικόνα 12 Ορισμός Logs στον κώδικα

Problems

Με το εργαλείο problems μπορούμε να διακρίνουμε και να εντοπίσουμε εύκολα και γρήγορα όλα τα λάθη και τις επισημάνσεις που έχει ο κώδικάς μας. Όπως βλέπουμε και στην παρακάτω εικόνα μας επισημάνει με κίτρινο τις προειδοποιήσεις, με πράσινο τα σωστά και με κόκκινο τα προβλήματα - λάθη που έχει η εφαρμογή.



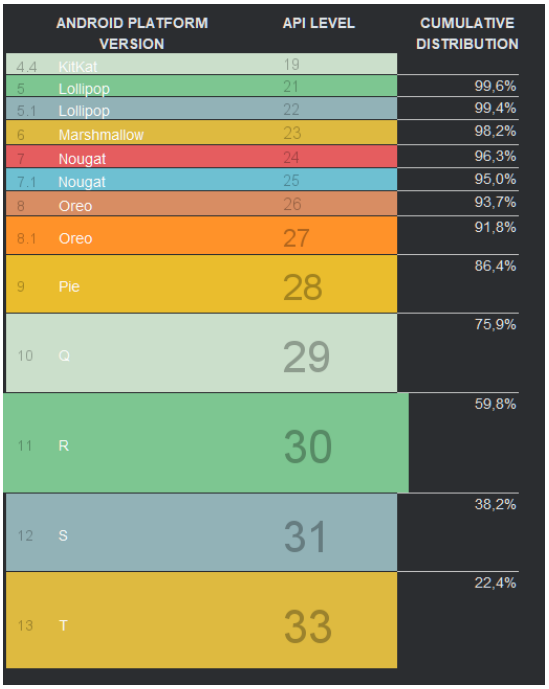
Εικόνα 13 Εργαλείο Problems

Android Virtual Device

Παρόλο που η προεπισκόπηση του Android Studio μας επιτρέπει να δούμε τη διάταξη που σχεδιάζουμε, για να δοκιμάσουμε εξ'ολοκλήρου ότι λειτουργεί η εφαρμογή μας, θα είναι απαραίτητο να συντάξουμε και να εκτελέσουμε ολόκληρο τον κώδικα. Μια εφαρμογή Android μπορεί να δοκιμαστεί εγκαθιστώντας και εκτελώντας την σε μια φυσική συσκευή ή σε ένα εικονικό συσκευής Android (AVD) στο περιβάλλον εξομοίωσης. Τα AVDs[7] είναι εξομοιωτές που επιτρέπουν τη δοκιμή εφαρμογών Android χωρίς την ανάγκη να εγκατασταθεί η εφαρμογή σε μια φυσική συσκευή βασισμένη σε Android.

Ένα AVD μπορεί να ρυθμιστεί προκειμένου να εξομοιώνει διάφορα χαρακτηριστικά υλικού, όπως το μέγεθος της οθόνης ή τη χωρητικότητα της μνήμης. Ακόμη, μπορούν να προστεθούν υπηρεσίες όπως η κάμερα ή μία υποστήριξη GPS πλοήγησης. Πολλά πρότυπα εξομοιωτών είναι εγκατεστημένα ως μέρος της τυπικής εγκατάστασης του Android Studio, επιτρέποντας τη ρύθμιση των AVDs για διάφορες συσκευές. Προσαρμοσμένες ρυθμίσεις όπως ο τύπος του επεξεργαστή, η χωρητικότητα της μνήμης και το μέγεθος και η πυκνότητα των pixel της οθόνης μπορούν να δημιουργήσουν οποιαδήποτε φυσική συσκευή Android.

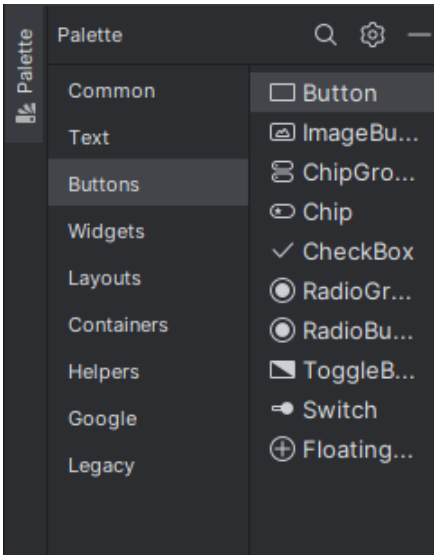
Για την υλοποίηση της εφαρμογής δημιουργήσαμε ένα από τα πρότυπα Android Virtual Device της κατηγορίας τηλεφώνου το Pixel 4. Τα χαρακτηρίστηκα της οθόνης από το Pixel 4 ήταν το μέγεθος 5,7”, ανάλυση οθόνης 1080*2080 και πυκνότητα 440dpi. Από την προτεινόμενη έκδοση συστήματος διαλέξαμε την R με API Level 30, τύπο Google Play, έκδοση android 11.0 και ABI x86. Σημαντικό ρόλο σε αυτή την επιλογή είχε το ποσοστό της τάξεως του 59,8% της χρήσης παρόμοιων κινητών τηλεφώνων από τους χρήστες παγκοσμίως.



Εικόνα 14 Ποσοστό χρήσης εκδόσεων Android

Palette

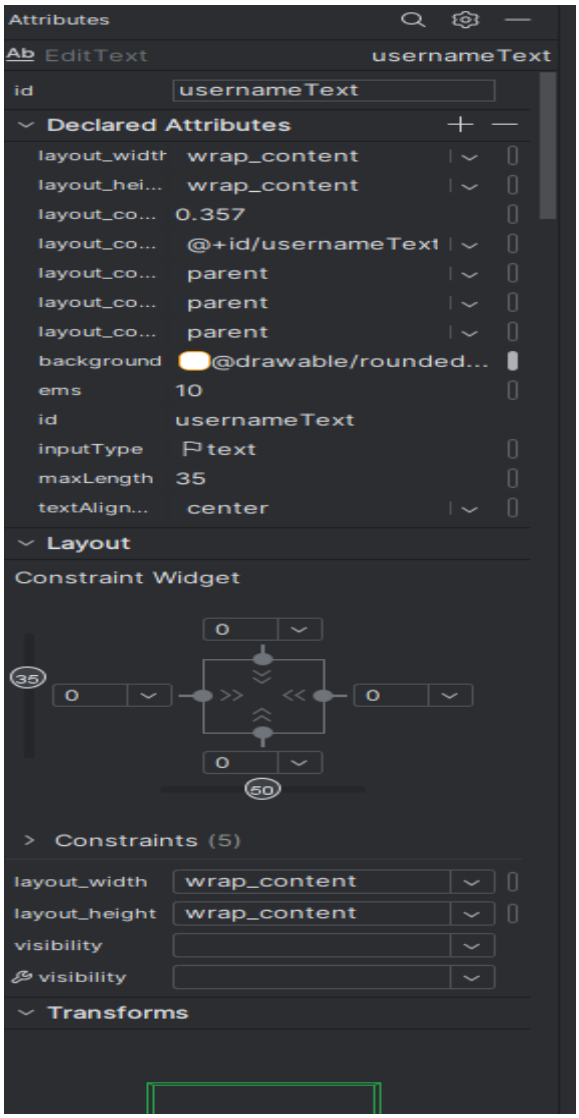
Μέσα στο UI View[24] μπορούμε να κάνουμε χρήση της τεχνικής drag and drop, ενώ από την καρτέλα palette μπορούμε να πάρουμε διάφορα γραφικά στοιχεία.



Εικόνα 15 Καρτέλα Palette

Attributes

Στη συνέχεια με την καρτέλα attributes μπορούμε να επεξεργαστούμε και να προγραμματίσουμε όλα τα γραφικά στοιχεία όπως επιθυμούμε.



Εικόνα 16 Καρτέλα Attributes

Kotlin

Εικόνα 17 Logo γλώσσας Kotlin

Η Kotlin[25][28] είναι μια στατική, μοντέρνου τύπου, αντικειμενοστραφής γλώσσα προγραμματισμού σχεδιασμένη από την ομάδα του Jet Brains. Η εταιρία Jet Brains εδρεύει στην Αγία Πετρούπολη της Ρωσίας, απ' όπου πήρε και το όνομα της η γλώσσα προγραμματισμού Kotlin, δηλαδή από το τοπικό νησί που βρίσκεται εκεί. Εκτελείται πάνω στην εικονική μηχανή της Java (JVM) και είναι συμβατή με την Java και το λειτουργικό σύστημα Android. Το 2011 έκανε την πρώτη της εμφάνιση, ως μια αναπτυσσόμενη γλώσσα ανοικτού κώδικα με την πρώτη επίσημη έκδοση να δημοσιεύετε το 2016. Το 2019 ανακοινώθηκε από την Google ως επίσημη πρώτη γλώσσα για το Android και αυτή τη στιγμή βρίσκετε στην έκδοση v1.9.23

Εικόνα 18 Έκδοση Kotlin

Είναι μια εκφραστική και συνοπτική γλώσσα προγραμματισμού που μειώνει τα κοινά λάθη και ενσωματώνετε εύκολα σε ήδη υπάρχον κώδικα. Επιτρέπει την ανάπτυξη εφαρμογών όχι μόνο για Android αλλά και για IOS, backend, desktop και Web εφαρμογές[26]. Έχει σχεδιαστεί για να αποτελέσει μια καλύτερη εκδοχή της Java. Εμπνευσμένη από πολλές γλώσσες προγραμματισμού όπως η C# και η Swift έχει σκοπό να πάρει τα δυνατά τους χαρακτηριστικά και να μηδενίσει τα λάθη τους.

Με αυτό το τρόπο ανέβασε το επίπεδο ανάπτυξης εφαρμογών με την ασφάλεια που παρέχει, την απόδοση του προγραμματιστή και την ποιότητα του κώδικα.

Γενικά θα την περιγράφαμε ως μια ασφαλής, εκφραστική, ευέλικτη και φιλική προς την χρήση εργαλείων γλώσσα προγραμματισμού, που έχει άμεση σύνδεση με την Java και την JavaScript. Ένα σημαντικό χαρακτηριστικό της Kotlin είναι ότι δεν είναι απαραίτητη η χρήση του ερωτηματικού στο τέλος κάθε εντολής. Άλλα κύρια χαρακτηριστικά που καθιστούν την Kotlin μοναδική είναι η διαλειτουργικότητα με τη Java που επιτρέπει τον κώδικα Kotlin να ενσωματώνεται εύκολα σε κώδικα Java και αντίστροφα. Επιπροσθέτως, έχει τη δυνατότητα χρήσης των Data Classes που απλοποιούν τη δημιουργία Getters και Setters, ενώ η Smart Cast επιτρέπει σε ένα αντικείμενο να αποκτήσει αυτόματα τον επιθυμητό τύπο. Ακόμη, περιλαμβάνει το null safety που προστατεύει από τα null exceptions, καθώς και τα named και default arguments τα οποία βελτιώνουν την ευαναγνωστηκότητα και την προκαθορισμένη τιμή παραμέτρων αντίστοιχα. Επίσης, οι Extension Functions προσθέτουν επιπλέον λειτουργικότητα σε υπάρχοντα στοιχεία ή κλάσεις, ενώ οι High Order Functions δέχονται ή επιστρέφουν συναρτήσεις ως παραμέτρους. Τέλος, υπάρχει η δυνατότητα σε μία συνάρτηση να επιστρέψει πολλές τιμές ταυτόχρονα.

Για όλους αυτούς τους λόγους πολλές επιχειρήσεις έχουν υιοθετήσει την Kotlin ως κύρια γλώσσα προγραμματισμού για την ανάπτυξη και την βελτίωση τους. Εκτός από την Jet Brains και την Google που είναι ιδρυτικά μέλη, την έχουν σαν κύρια γλώσσα προγραμματισμού και έχουν προσχωρήσει σε αυτή μεγάλες εταιρίες ως silver members εταιρίες όπως η Gradle, το Shopify και το Touch lab. Η Kotlin έχει κάθε μήνα πάνω από 710.000 ενεργούς προγραμματιστές, στο GitHub την έχουν σαν κύρια γλώσσα πάνω από 1,2 εκατομμύρια χρήστες και την διδάσκουν στα 32 από τα 100 καλύτερα πανεπιστήμια παγκοσμίως.

SQL

Εικόνα 19 Logo SQLite

Το SQLite[29][30] είναι ένα ενσωματωμένο σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων ανοιχτού κώδικα. Τα περισσότερα σχεσιακά συστήματα βάσεων δεδομένων όπως το Oracle, το SQL Server και το MySQL είναι αυτόνομοι διακομιστές που λειτουργούν ανεξάρτητα και συνεργάζονται με εφαρμογές που απαιτούν πρόσβαση στη βάση δεδομένων. Το SQLite ονομάζεται ενσωματωμένο επειδή είναι μια βιβλιοθήκη της γλώσσας προγραμματισμού C που υλοποιεί μία μικρή, γρήγορη, αυτόνομη, αξιόπιστη και πλήρης εξοπλισμένη βάση δεδομένων SQL.

Το project της SQLite ξεκίνησε στις 9 Μαΐου του 2000. Μπορεί να χρησιμοποιηθεί χωρίς την σύνδεση του χρήστη στο διαδίκτυο και δεν είναι μια ξεχωριστή διεργασία του προγράμματος, αλλά ενσωματωμένη σε αυτό. Όλες οι λειτουργίες της βάσης δεδομένων χειρίζονται εσωτερικά μέσα στην εφαρμογή μέσω κλήσεων στις λειτουργίες της βιβλιοθήκης SQLite. Επίσης, δημιουργεί μια βάση δεδομένων τοπικά στη συσκευή με αποτέλεσμα να αποθηκεύει και να επεξεργάζεται τα δεδομένα που καταχωρεί ο χρήστης εκεί. Λόγω της εύκολης χρήσης και του μικρού χώρου που καταλαμβάνει, είναι μια δημοφιλής επιλογή ως ενσωματωμένη βάση δεδομένων για τοπική αποθήκευση. Το SQLite μπορεί να επικοινωνήσει με αρκετές γλώσσες προγραμματισμού και χρησιμοποιεί τα ίδια query με την MySQL.

Το SQLite είναι μια δημοφιλής, συμπαγής και διασυνοριακή βάση δεδομένων SQL που περιλαμβάνει πολλαπλούς πίνακες, δείκτες, εναύσματα και προβολές σε ένα αρχείο δίσκου. Η μορφή αρχείου είναι cross-platform, επιτρέποντας τη μεταφορά δεδομένων μεταξύ συστημάτων 32-bit και 64-bit και διαφορετικών αρχιτεκτονικών. Αναγνωρίζεται ως συνιστάμενη μορφή αποθήκευσης από τη βιβλιοθήκη του Κογκρέσου των ΗΠΑ και θεωρείται αντικατάσταση του fopen. Παρά το συμπαγές μέγεθός της, το SQLite προσφέρει καλή απόδοση και αξιοπιστία, με εκατομμύρια δοκιμαστικές περιπτώσεις SQL και πλήρη δοκιμή του πηγαίου κώδικα. Ενώ η χρήση της μνήμης επηρεάζει την ταχύτητα, η απόδοση παραμένει αξιόπιστη ακόμη και σε περιβάλλοντα χαμηλής μνήμης. Υποστηρίζεται από μια διεθνή ομάδα προγραμματιστών, που συνεχίζει να επεκτείνει τις δυνατότητες της και να βελτιώνει την απόδοση και την αξιοπιστία της βάσης δεδομένων.

Ένα από τα μεγαλύτερα χαρακτηριστικά της είναι η μη παραμετροποίηση της πριν την χρήση και η ελεύθερη μεταφορά της μεταξύ διαφορετικών συστημάτων. Η βάση δεδομένων SQLite διακρίνεται για τη χρησιμοποίηση ελάχιστων πόρων μνήμης κατά την εκτέλεσή της, ενισχύοντας την ενσωμάτωσή της σε συστήματα με περιορισμένη διαθεσιμότητα πόρων. Επιπλέον, διαθέτει μηχανισμούς κρυπτογράφησης για την αποτελεσματική προστασία των δεδομένων, εξασφαλίζοντας την ασφαλή αποθήκευση ευαίσθητων πληροφοριών. Σε ό,τι αφορά την ακεραιότητα και τη σταθερότητα των δεδομένων, παρέχει προηγμένες τεχνολογίες συναλλαγών, διασφαλίζοντας την Ατομικότητα, τη Συνέπεια, την Απομόνωση και τη Μόνιμη αποθήκευση (ACID). Επιπλέον, η αρχιτεκτονική του είναι ειδικά σχεδιασμένη για την αυτόματη διαχείριση σφαλμάτων, όπως αποτυχίες εκχώρησης μνήμης, ενώ παρέχει ευελιξία στην ενσωμάτωσή του σε εφαρμογές που υλοποιούνται σε διάφορες γλώσσες προγραμματισμού, όπως C, C++, Java και Python. Ως ανοιχτού κώδικα λογισμικό, ο πηγαίος κώδικας του είναι διαθέσιμος για εκτεταμένη ανάλυση και προσαρμογή, ενώ υπάρχουν εταιρείες που προσφέρουν

επαγγελματική υποστήριξη για την αξιοποίηση και διαχείριση της βάσης δεδομένων SQLite.

XML

Η Extensible Markup Language (XML)[31][32][33] είναι μια διευρυμένη γλώσσα σήμανσης που περιέχει ένα σύνολο κανόνων για την ηλεκτρονική κωδικοποίηση κειμένων, η οποία προέκυψε από το Standard Generalized Markup Language (SGML - ISO 8879). Αρχικά, αναπτύχθηκε με σκοπό να ανταποκριθεί στις απαιτήσεις των μεγάλης κλίμακας ηλεκτρονικών εκδόσεων, ωστόσο έχει αποκτήσει σημαντική θέση και χρήση στη διεθνή ανταλλαγή δεδομένων στον Παγκόσμιο Ιστό και σε άλλες πλατφόρμες.

Δημιουργήθηκε από τον διεθνής οργανισμό προτύπων W3C (World Wide Web Consortium), αλλά και από άλλες προδιαγραφές ανοιχτών προτύπων. Περιέχει κανόνες για τον ορισμό οποιωνδήποτε δεδομένων και δεν μπορεί να εκτελέσει υπολογιστικές λειτουργίες από μόνη της, όπως άλλες γλώσσες προγραμματισμού. Μπορούμε να αποθηκεύσουμε και να μεταφέρουμε δεδομένα διασφαλίζοντας την ακεραιότητα τους και να ορίσουμε τα δικά μας tag, ανάλογα με τις ανάγκες του κώδικα μας, που σε άλλες γλώσσες όπως η HTML (HyperText Markup Language) είναι προκαθορισμένα. Είναι μια μορφοποίηση δεδομένων κειμένου, με ισχυρή υποστήριξη της Unicode για όλες τις γλώσσες. Αν και σχεδιάστηκε για την μορφοποίηση κειμένων χρησιμοποιείται και για την αναπαράσταση δομών δεδομένων.

Βιβλιοθήκες

Στις γλώσσες προγραμματισμού με την λέξη Βιβλιοθήκες, εννοούμε ένα σύνολο από έτοιμους και λειτουργικούς κώδικες. Σχεδιάστηκαν με σκοπό να παρέχουν συγκεκριμένες λειτουργίες που μπορούν να επιτευχθούν χωρίς ο προγραμματιστής να γράψει κώδικα. Με τις βιβλιοθήκες εξοικονομείται χρόνος και προσπάθεια καθώς αποφεύγεται η προγραμματιστική υλοποίηση κάποιας που χρειάζεται για το πρόγραμμα.

Οι βιβλιοθήκες που έχουν χρησιμοποιηθεί στην εφαρμογή είναι οι εξής:

Βιβλιοθήκη AndroidX Lifecycle

- implementation("androidx.lifecycle:lifecycle-viewmodel-ktx:2.6.2")

Είναι μια εξάρτηση της βιβλιοθήκης AndroidX Lifecycle[8] που υποστηρίζει τη διαχείριση του Lifecycle σε Android εφαρμογές. Συγκεκριμένα στη δημιουργία και τη διαχείριση του ViewModel σε συνδυασμό με την Kotlin.

- implementation("androidx.lifecycle:lifecycle-livedata-ktx:2.6.2")

Είναι μια εξάρτηση από τη βιβλιοθήκη του AndroidX Lifecycle που παρέχει υποστήριξη για τη διαχείριση δεδομένων τύπου LiveData σε συνδυασμό με την Kotlin

Βιβλιοθήκη SQLite

- implementation("androidx.sqlite:sqlite:2.4.0")

Είναι μία εξάρτηση της βιβλιοθήκης SQLite[15] που παρέχει υποστήριξη στην διαχείριση της βάσης δεδομένων. Με αυτή παρέχονται λειτουργίες όπως η δημιουργία, η ανάγνωση, η ενημέρωση και η διαγραφή δεδομένων.

Βιβλιοθήκη Core

- implementation("androidx.core:core-ktx:1.9.0")

Είναι μια εξάρτηση της βιβλιοθήκης Core[9] που σου επιτρέπει την διαχείριση των κουμπιών, των ρυθμίσεων, τον έλεγχο της πρόσβασης στο δίκτυο και άλλων βασικών στοιχείων της εφαρμογής.

Βιβλιοθήκη AppCompat

- implementation("androidx.appcompat:appcompat:1.6.1")

Είναι μια εξάρτηση της βιβλιοθήκης AppCompat[10] που σου προσφέρει την συμβατότητα της εφαρμογής σε παλαιότερες εκδόσεις του Android.

Βιβλιοθήκη Material Design

- implementation("com.google.android.material:material:1.10.0")

Η βιβλιοθήκη Material Design[11] προσφέρει τα δικά της στυλιστικά στοιχεία και σχεδιαστικά πρότυπα που συμβαδίζουν με τις οδηγίες του υλικού σχεδιασμού της Google.

Βιβλιοθήκη ConstraintLayout

- implementation("androidx.constraintlayout:constraintlayout:2.1.4")

Η ConstraintLayout[12] προσφέρει έναν ισχυρό και ευέλικτο τρόπο διάταξης των στοιχείων στο layout, επιτρέποντας τη δημιουργία πολύπλοκων διατάξεων που προσαρμόζονται στις ανάγκες της εφαρμογής.

Βιβλιοθήκη Legacy

- implementation("androidx.legacy:legacy-support-v4:1.0.0")

Η Legacy[18] προσφέρει υποστήριξη σε παλιότερες εκδόσεις android.

Βιβλιοθήκη RecyclerView

- implementation("androidx.recyclerview:recyclerview:1.3.0")

Με τη Βιβλιοθήκη RecyclerView[14] μπορεί ο προγραμματιστής να εμφανίσει μεγάλου όγκου δεδομένα σε λίστες ή πίνακες.

Βιβλιοθήκη Junit

- testImplementation("junit:junit:4.13.2")

Είναι μια βοηθητική βιβλιοθήκη για δοκιμαστικές εκτελέσεις του κώδικα της εφαρμογής για την εξασφάλιση της σωστής λειτουργίας του.[17]

Βιβλιοθήκη Test Extension

- androidTestImplementation("androidx.test.ext:junit:1.1.5")

Είναι μια βοηθητική βιβλιοθήκη για την ομαλή λειτουργία της βιβλιοθήκης Junit.[16]

Βιβλιοθήκη Espresso

- androidTestImplementation("androidx.test.espresso:espresso-core:3.5.1")

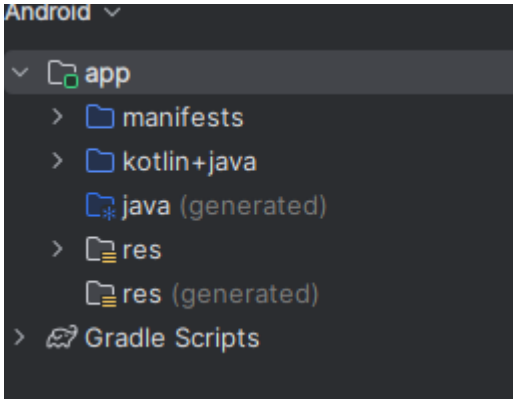
Εκτελούνται αυτοματοποιημένες δοκιμές στις εφαρμογές για τον έλεγχο της αλληλεπίδρασης του χρήστη και τη σωστή λειτουργεία της.[13]

Εισαγωγή στην εφαρμογή MFC

Η Maintenance Fees Calculator (MFC) είναι μία απλή και εύχρηστη για τον υπολογισμό των κοινοχρήστων εφαρμογή. Μπορεί να χρησιμοποιηθεί από τον οποιοδήποτε μόνο με την χρήση ενός έξυπνου τηλεφώνου και τη γνώση των βασικών στοιχείων της οικοδομής και των ένοικων που μένουν σε αυτή. Μερικά από αυτά τα στοιχεία είναι τα τετραγωνικά μέτρα, ο αριθμός των διαμερισμάτων και ένα τηλέφωνο επικοινωνίας από τους ενοίκους. Με αυτό το τρόπο και καταχωρώντας ο χρήστης κάθε μήνα τα έξοδα της πολυκατοικίας, η εφαρμογή μπορεί να υπολογίσει το συνολικό ποσό εξόδων που αντιστοιχεί σε κάθε ένοικο.

Περιγραφή δομής της εφαρμογής

Η MFC αποτελείτε από τέσσερεις κύριους φακέλους και αρχεία, τα manifest, τα Kotlin, τα resources και τα grandle scripts.



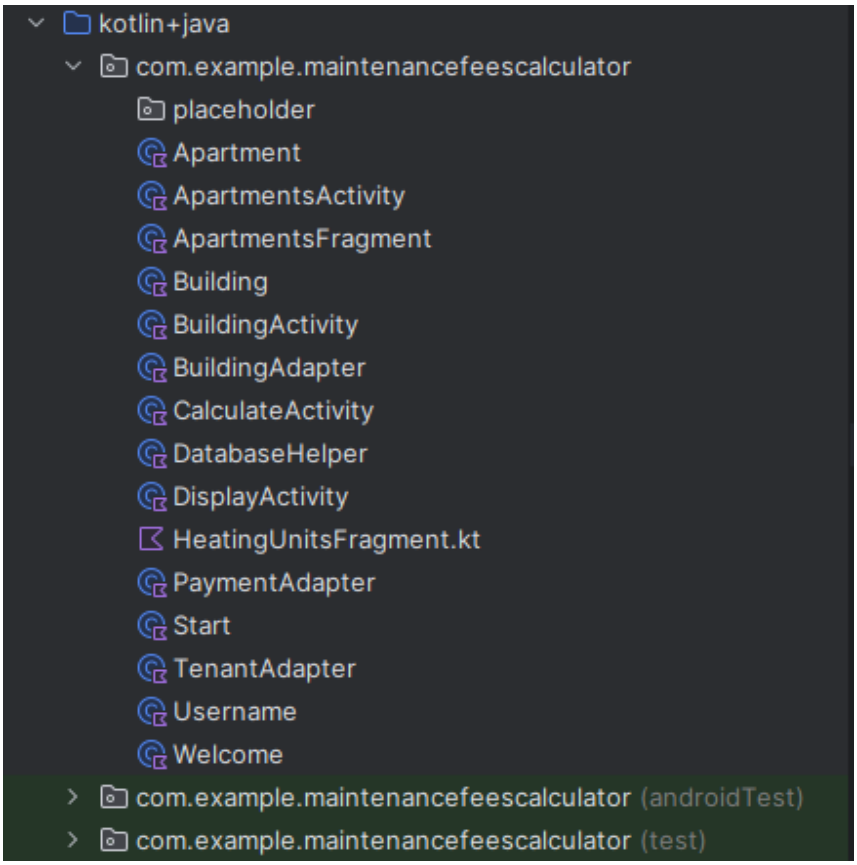
Εικόνα 20 Κύρια αρχεία της εφαρμογής

Τα manifest είναι απαραίτητα αρχεία xml που περιγράφουν τις βασικές πληροφορίες όπως το όνομα και η έκδοση της εφαρμογής. Μέσα σε αυτά τα αρχεία θα βρούμε και τα δικαιώματα που θα ζητήσει η εφαρμογή από το λογισμικό του

κινητού τηλεφώνου, όπως σύνδεση με το διαδίκτυο, την κάμερα, το GPS και ότι άλλο είναι απαραίτητο για τη σωστή λειτουργεία της. Ακόμα, θα βρούμε τις δηλώσεις των activity, των υπηρεσιών και των διακομιστών που παρέχει η εφαρμογή.

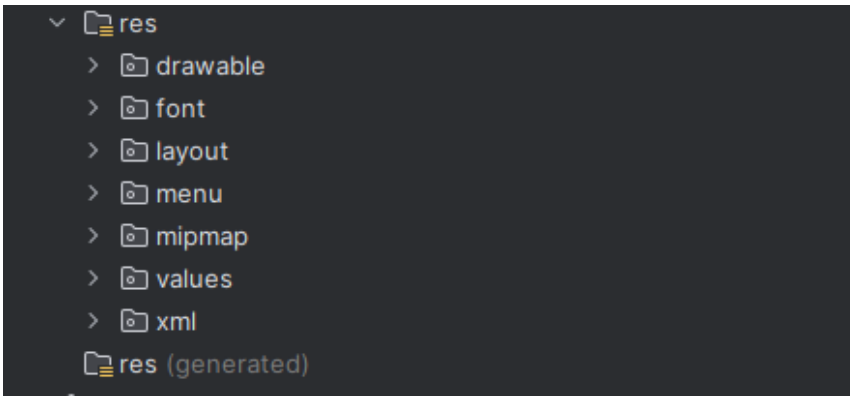
Ο φάκελος Kotlin περιλαμβάνει αρχεία Kotlin που σε αυτά έχει γραφτεί ο κύριος κώδικας της εφαρμογής. Αυτός ο φάκελος περιέχει τον κώδικα από τα activities, τα fragment, τα adapter και τη βάση δεδομένων που έχουν δημιουργήσει την εφαρμογή. Όσον αφορά τα activities, πρόκειται για αρχεία που το καθένα αντιπροσωπεύει μια οθόνη UI (User Interface) και σε αυτά γράφουμε κώδικα που ελέγχει την συμπεριφορά της εφαρμογής ανάλογα με την ενέργεια του χρήστη. Επίσης, οι εφαρμογές Android δημιουργούνται συνδυάζοντας ένα ή περισσότερα activities.

Στον κώδικα Kotlin θα βρούμε δηλώσεις import που χρησιμοποιούνται για την εισαγωγή κλάσεων και μεθόδων, δηλώσεις των παραμέτρων, functions και όλη την διεργασία που θέλουμε να κάνει η εφαρμογή κατά την εκτέλεση της. Επιπλέον, τα fragment είναι αρχεία με επαναχρησιμοποιούμενο κώδικα που μπορούν να ενσωματωθούν ή να αντικατασταθούν μέσα σε ένα activity. Τα κύρια χαρακτηριστικά του είναι η επαναχρησιμοποίηση, η σύνθετη διεπαφή όπου ένα activity μπορεί να αλληλοεπιδρά με πολλά fragment και η διαχείριση των δεδομένων και των UI καταστάσεων. Τέλος, τα adapter είναι αρχεία που μας επιτρέπουν την εμφάνιση δεδομένων μέσα σε ένα view, recyclerview ή listview. Η βάση δεδομένων είναι και αυτή ένα αρχείο Kotlin που περιέχει μια κλάση DatabaseHelper. Μέσα σε αυτή την κλάση δηλώνουμε την βάση δεδομένων με όλους τους πίνακες, τις στήλες και τις γραμμές που θέλουμε να δημιουργήσουμε. Σε αυτό το αρχείο χρησιμοποιούμε εντολές SQL που μπορούμε να αποθηκεύσουμε δεδομένα και να κάνουμε πράξεις με αυτά.



Εικόνα 21 Φάκελος Kotlin

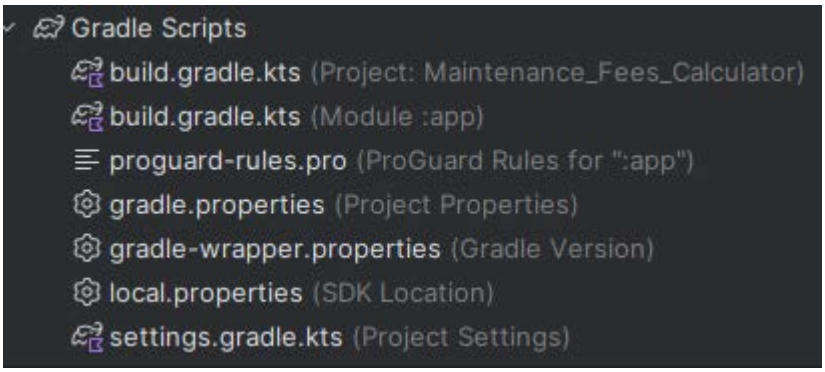
Στον φάκελο resources θα βρούμε τους υποφάκελους drawable, font, layout, menu, mipmap, values και xml.



Εικόνα 22 Φάκελος resources

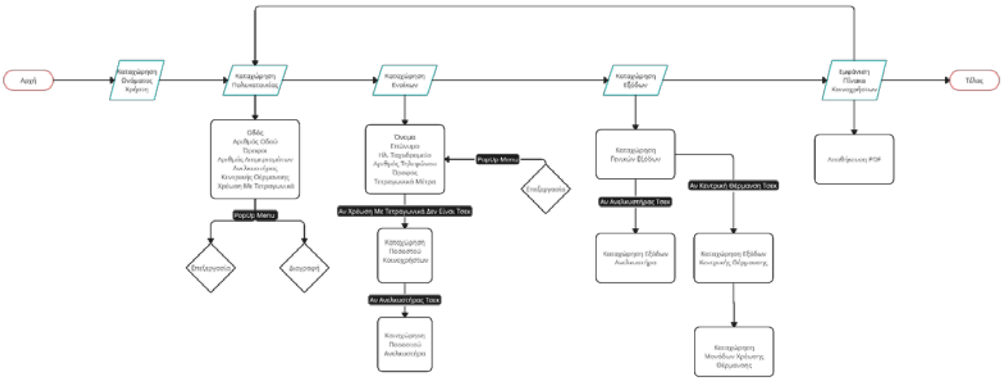
Στα drawable αρχεία θα βρούμε τις εικόνες που έχουμε χρησιμοποιήσει για το background, τα layouts και τα fab (Floating Action Button). Περιέχονται και αρχεία xml με την διαμόρφωση των περιγραμμάτων και των ορίων των κελιών. Στον φάκελο font αποθηκεύεται η γραμματοσειρά, στο menu το μενού και mipmap το εικονίδιο του launcher που χρησιμοποιεί η εφαρμογή. Στα values περιέχονται αρχεία για την διαχείριση πόρων όπως τα χρώματα, τις διαστάσεις για τις διάφορες οθόνες, τις συμβολοσειρές και το στυλ που χρησιμοποιεί για τα UI στοιχεία της η εφαρμογή. Στον υποφάκελο layout βρίσκουμε αρχεία xml που ορίζουν τη δομή των UI στοιχείων των activity, fragment και items που έχουμε δημιουργήσει στον φάκελο Kotlin. Στον φάκελο xml υπάρχουν γενικά αρχεία για τις ρυθμίσεις της εφαρμογής.

Τα gradle scripts είναι αρχεία που δημιουργούνται αυτόματα κατά την δημιουργία ενός project και καθορίζουν την δημιουργία της εφαρμογής. Περιέχει δύο βασικά αρχεία, το build gradle project και το build gradle module. Το καθένα από αυτά τα αρχεία είναι διαφορετικές εκδοχές της εφαρμογής που μπορούν να εκτελεστούν και να επιδιορθωθούν ξεχωριστά η μια από την άλλη. Το build gradle project περιέχει επιλογές διαμόρφωσης που είναι κοινές για όλα τα modules του project. Αντίθετα το build gradle module διαθέτει ρυθμίσεις για την κατασκευή ενός module που διαθέτει πληροφορίες όπως το όνομα, στις εκδόσεις android που μπορεί να τρέξει, και τα plugins της εφαρμογής.



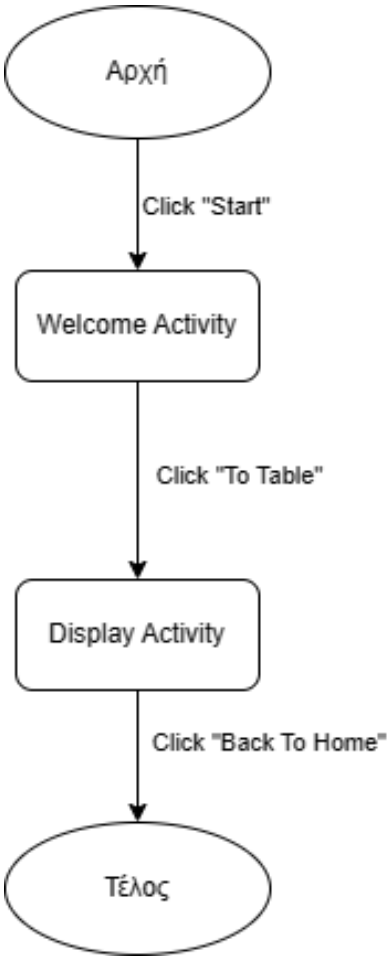
Εικόνα 23 Gradle Scripts

Παρακάτω σας παρουσιάζουμε με ένα διάγραμμα όλες τις δυνατότητες που παρέχει η εφαρμογή στον χρήστη.



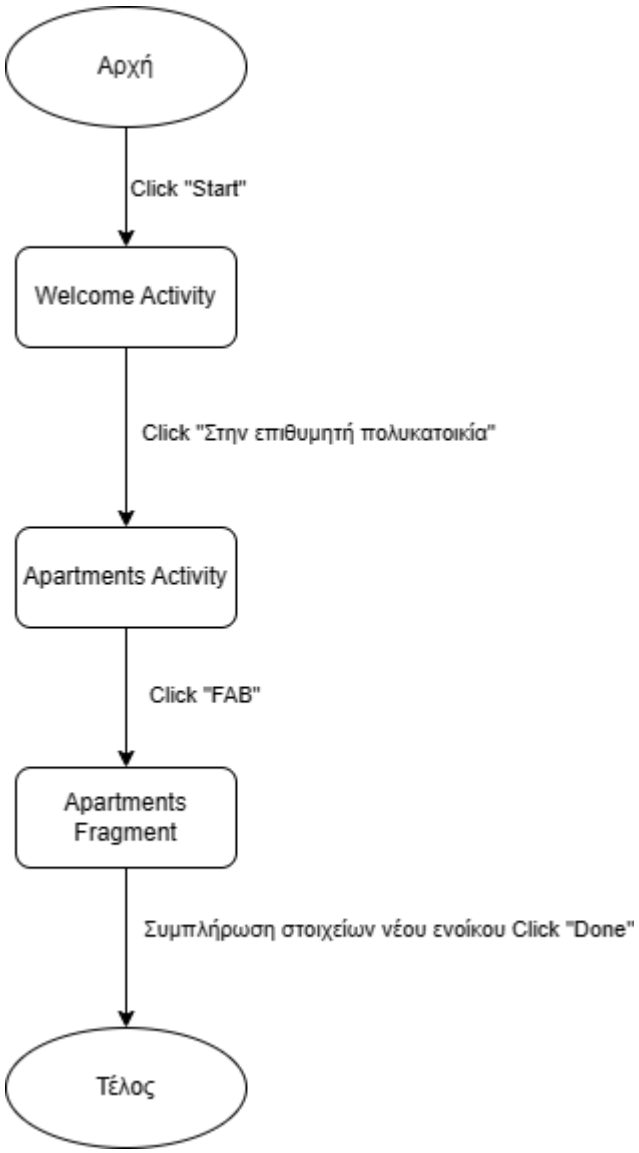
Εικόνα 24 Διάγραμμα ανάλυσης δυνατοτήτων του χρήστη

Στο παρακάτω Use Case Diagram φαίνεται η πορεία που ακολουθεί ο χρήστης στην εφαρμογή, ώστε να μπορέσει να ανατρέξει άμεσα σε παλαιότερες καταχωρήσεις μέσω ενός συνολικού πίνακα, για κάθε πολυκατοικία που διαχειρίζεται.



Εικόνα 25 UML προβολής πίνακα

Στο παρακάτω Use Case Diagram φαίνεται η πορεία που ακολουθεί ο χρήστης στην εφαρμογή, για την εισαγωγή ενός νέου ενοίκου σε μια πολυκατοικία που διαχειρίζεται.



Εικόνα 26 UML εισαγωγής νέου ενοίκου

Περιγραφή υλοποίησης μεθόδων

On Create

Η On Create[19] μέθοδος αρχικοποιεί το activity και ορίζει το περιεχόμενο του από το layout αρχείο που είναι απαραίτητα για τη σωστή λειτουργία του activity.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_start)

    val start = findViewById<Button>(R.id.start)
    start.setOnClickListener { it: View!
        val Intent = Intent( packageContext: this,Username::class.java)
        startActivity(Intent)
    }
}
```

Εικόνα 27 On Create

Όπως βλέπουμε στην παραπάνω εικόνα υπάρχει μία μέθοδος On Create. Με την γραμμή κώδικα, override fun onCreate(savedInstanceState: Bundle?) {, δηλώνουμε την μέθοδο που καλείτε με την δημιουργία του πρώτου activity της εφαρμογής. Στη συνέχεια με την εντολή, super.onCreate(savedInstanceState), καλεί την OnCreate και την αρχικοποιεί. Με την εντολή, setContentView(R.layout.activity_start), ορίζει το περιεχόμενο της activity χρησιμοποιώντας το Layout αρχείο (activity_start), που περιέχει τα UI στοιχεία που θα εμφανιστούν στην οθόνη του χρήστη.

On Upgrade

Με την μέθοδο On Upgrade μπορούμε να ανανεώσουμε τα παλιά στοιχεία με νέα.

```
override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
    db.execSQL(SQL_DELETE_USER_ENTRIES)
    db.execSQL(SQL_DELETE_ADDRESS_ENTRIES)
    db.execSQL(SQL_DELETE_APARTMENTS_ENTRIES)
    db.execSQL(SQL_DELETE_TOTAL_PAYMENTS_ENTRIES)
    db.execSQL(SQL_DELETE_TOTAL_EXPENSES_ENTRIES)
    onCreate(db)
}
```

Εικόνα 28 On Upgrade

Όπως βλέπουμε στην εικόνα, με την μέθοδο On Upgrade μπορούμε να διαγράψουμε την παλιά έκδοση της βάσης δεδομένων και να την ανανεώσουμε με μία νέα έκδοση που θα έχει καταχωρήσει τα νέα στοιχεία ο χρήστης.

On Click Listener

Με τον listener OnClick[21] προγραμματίζουμε την δραστηριότητα που θα κάνει το κουμπί όταν το πατήσει ο χρήστης.

```
val start = findViewById<Button>(R.id.start)
start.setOnClickListener { it: View!
    val Intent = Intent( packageContext: this,Username::class.java)
    startActivity(Intent)
}
```

Εικόνα 29 On Click Listener

Στο παραπάνω παράδειγμα με την εντολή, val start = findViewById <Button> (R.id. start), δηλώνουμε στη μεταβλητή start το κουμπί με id start από την activity start. Με την εντολή, start.setOnClickListener {, ορίζουμε ένα Listener που θα εκτελεστεί με το πάτημα του κουμπιού. Με τον κώδικα, val Intent = Intent(this, Username::class.java), δημιουργούμε ένα Intent που το προσανατολίζουμε με την activity Username. Με το, startActivity(Intent), ξεκινάμε την activity Username η οποία εμφανίζετε στον χρήστη.

On Resume

Με την μέθοδο On Resume μπορούμε να κάνουμε ανανέωση μίας μεθόδου κάθε φορά που το activity γίνεται ορατό στον χρήστη, δηλαδή επιστρέφει στο παρασκήνιο.

```
override fun onResume() {
    super.onResume()


    updateBuildingList()
}
```

Εικόνα 30 On Resume

Παρατηρώντας το παραπάνω παράδειγμα, με την εντολή override fun onResume(), δηλώνουμε την μέθοδο η οποία καλείται κάθε φορά που η δραστηριότητα γίνεται ορατή στο χρήστη. Η εντολή, super.onResume(), είναι απαραίτητη για την σωστή λειτουργία της function, ενώ με την μέθοδο

updateBuildingList ανανεώνεται η λίστα των κτηρίων κάθε φορά που ο χρήστης πηγαίνει σε αυτό το activity.

Content View

Με την Content View μπορούμε να ορίσουμε το περιεχόμενο μιας μεταβλητής από δύο διαφορετικά activity

```
setContentView(R.layout.activity_username)
usernameEditText = findViewById(R.id.usernameText)
val submitButton = findViewById<Button>(R.id.submitButton)
submitButton.setOnClickListener { it: View!
    val username = usernameEditText.text.toString()

    if (username.isNotBlank()) {

        saveUsernameToDatabase(username)

        with(sharedPref.edit()) { this: SharedPreferences.Editor!
            putBoolean("USERNAME_SET", true)
            apply()
        }
        startNextActivity(username)
    }
}
```

Εικόνα 31 Content View

Βλέποντας το παράδειγμα, με την εντολή, setContentView(R.layout.activity _user name), ορίζουμε το περιεχόμενο της activity χρησιμοποιώντας το layout αρχείο activity_username. Με την εντολή, usernameEditText = findViewById(R.id.usernameText), καταχωρούμε την μεταβλητή από το usernameText στο usernameEditText της άλλης activity.

Are Fields Empty

Με την Are fields empty μπορούμε να ελέγξουμε αν ο χρήστης έχει καταχωρίσει τα στοιχεία σε όλα τα απαραίτητα πεδία για τη σωστή λειτουργεία της εφαρμογής.

```
private fun areFieldsEmpty(layout: ViewGroup): Boolean {
    var allFieldsFilled = false


    for (i in 0 ≤ until  < layout.childCount) {
        val childView = layout.getChildAt(i)


        if (childView is ViewGroup) {

            allFieldsFilled = areFieldsEmpty(childView)
            if (!allFieldsFilled) {

                return false
            }
        }

        if (childView is EditText) {

            if (childView.text.isNullOrEmpty()) {

                return false
            }
        }
    }
```

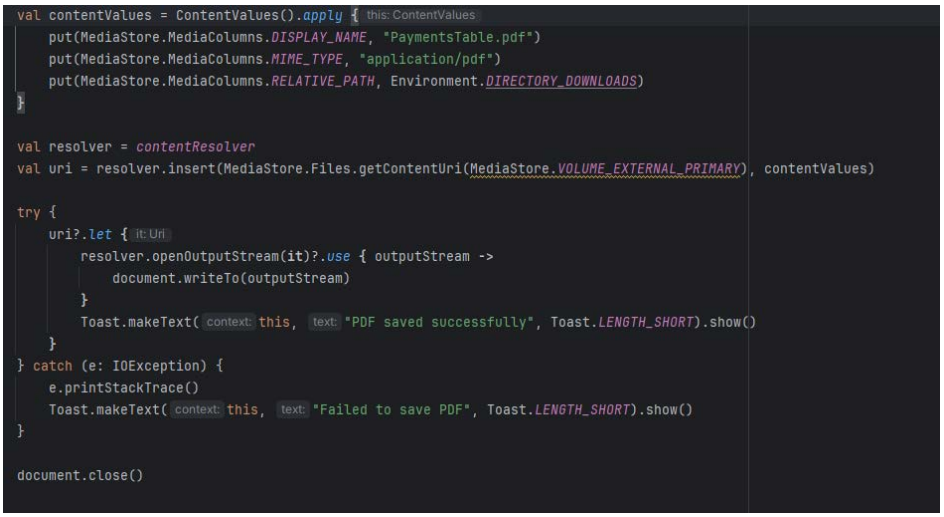
Εικόνα 32 Are Fields Empty

Generate PDF

Με την μέθοδο Generate pdf[20] δημιουργούμε ένα pdf στο οποίο έχουμε στοιχεία που τα παίρνουμε μέσα από ένα recycler view. Όπως βλέπουμε παρακάτω, παίρνουμε τα στοιχεία που θέλουμε από το recycler view και προγραμματίζουμε το pdf για το μέγεθος, το χρώμα και τα στοιχεία που θέλουμε να εμφανίζονται. Στη συνέχεια, ονομάζουμε το pdf και το αποθηκεύουμε σαν αρχείο pdf στα downloads του κινητού τηλεφώνου του χρήστη.

```
private fun generatePDF(startPosition: Int, endPosition: Int) {
    Log.d( tag: "PDF Generation",  msg: "Generating PDF document")
    val document = PdfDocument()

    val headerView = LayoutInflater.from( context: this).inflate(R.layout.item_payment_header,  root: null)
    headerView.measure(
        View.MeasureSpec.makeMeasureSpec(recyclerView.width, View.MeasureSpec.EXACTLY),
        View.MeasureSpec.makeMeasureSpec( size: 0, View.MeasureSpec.UNSPECIFIED))
    headerView.layout( l: 0,  t: 0, headerView.measuredWidth, headerView.measuredHeight)

    val pageInfo = PdfDocument.PageInfo.Builder(headerView.width, headerView.height,  pageNumber: 1).create()
    val page = document.startPage(pageInfo)
    val canvas: Canvas = page.canvas
    canvas.drawColor(Color.BLACK)
    headerView.draw(canvas)
    document.finishPage(page)

    for (i in startPosition ≤ until  < endPosition) {
        val viewHolder = recyclerView.findViewHolderForAdapterPosition(i)
        viewHolder?.itemView?.let { itemView ->
            Log.d( tag: "PDF Generation",  msg: "Drawing page $i")
            val pageInfo = PdfDocument.PageInfo.Builder(itemView.width, itemView.height, i).setContentRect(
                Rect( left: 0,  top: 0, itemView.width, itemView.height)
            ).create()
            val backgroundColor = Color.rgb( red: 50,  green: 50,  blue: 50)
            val page = document.startPage(pageInfo)
            val canvas: Canvas = page.canvas
            canvas.drawColor(backgroundColor)
            itemView.draw(canvas)
            document.finishPage(page)
        }
    }
```

Εικόνα 33 Generate PDF



```
val contentValues = ContentValues().apply { this: ContentValues
    put(MediaStore.MediaColumns.DISPLAY_NAME, "PaymentsTable.pdf")
    put(MediaStore.MediaColumns.MIME_TYPE, "application/pdf")
    put(MediaStore.MediaColumns.RELATIVE_PATH, Environment.DIRECTORY_DOWNLOADS)
}

val resolver = contentResolver
val uri = resolver.insert(MediaStore.Files.getContentUri(MediaStore.VOLUME_EXTERNAL_PRIMARY), contentValues)

try {
    uri?.let { it: Uri
        resolver.openOutputStream(it)?.use { outputStream ->
            document.writeTo(outputStream)
        }
        Toast.makeText( context: this,  text: "PDF saved successfully", Toast.LENGTH_SHORT).show()
    }
} catch (e: IOException) {
    e.printStackTrace()
    Toast.makeText( context: this,  text: "Failed to save PDF", Toast.LENGTH_SHORT).show()
}

document.close()
```

Εικόνα 34 Download PDF

Save Expenses to Data Base

Με την μέθοδο save expenses to data base μπορούμε να δούμε πως αποθηκεύονται τα στοιχεία που καταχωρεί ο χρήστης στη βάση δεδομένων που έχουμε φτιάξει.

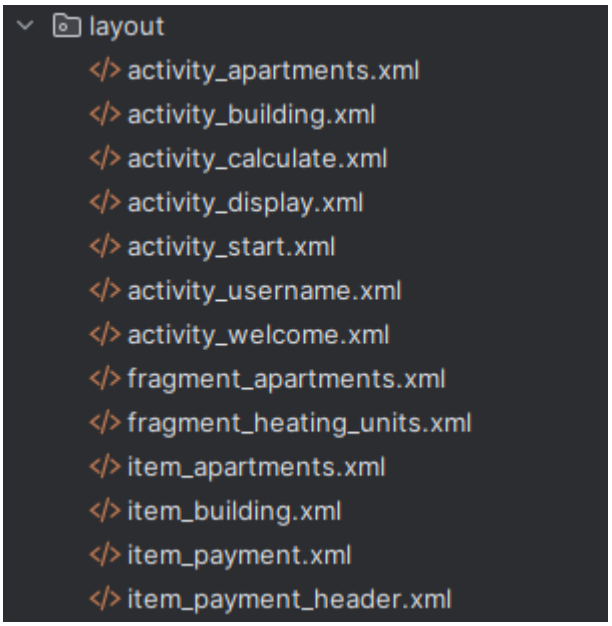
```
private fun saveExpensesToDatabase(buildingId: Long) {
    val totalExpensesEditText = findViewById<EditText>(R.id.inputTotal)
    val elevatorExpensesEditText = findViewById<EditText>(R.id.inputElevator)
    val heatingExpensesEditText = findViewById<EditText>(R.id.inputHeating)

    val totalExpensesInput = totalExpensesEditText.text.toString().toFloatOrNull() ?: 0.0f
    val elevatorExpensesInput = elevatorExpensesEditText.text.toString().toFloatOrNull() ?: 0.0f
    val heatingExpensesInput = heatingExpensesEditText.text.toString().toFloatOrNull() ?: 0.0f
    val currentMonthYearInput = getCurrentMonthYear()

    val dbHelper = DatabaseHelper( context: this)
    val db = dbHelper.writableDatabase

    val values = ContentValues().apply { this: ContentValues
        put(DatabaseHelper.TotalExpensesEntry.COLUMN_NAME_GENERAL_EXPENSES, totalExpensesInput)
        put(DatabaseHelper.TotalExpensesEntry.COLUMN_NAME_ELEVATOR_EXPENSES, elevatorExpensesInput)
        put(DatabaseHelper.TotalExpensesEntry.COLUMN_NAME_HEATING_EXPENSES, heatingExpensesInput)
        put(DatabaseHelper.TotalExpensesEntry.COLUMN_NAME_MONTH_YEAR, currentMonthYearInput)
        put(DatabaseHelper.TotalExpensesEntry.COLUMN_NAME_BUILDING_ID, buildingId)

    }


    db.insert(DatabaseHelper.TotalExpensesEntry.TABLE_NAME,  nullColumnHack: null, values)

}
```

Εικόνα 35 Save to Data Base

Με τις εντολές, val totalExpensesEditText = findViewById<EditText>(R.id.inputTotal) και val totalExpensesInput = totalExpensesEditText.text.toString().toFloatOrNull() ?: 0.0f, καταχωρούμε το σύνολο των γενικών εξόδων που έχει προσθέσει ο χρήστης στο edit text με id inputTotal. Στη συνέχεια με τις εντολές val dbHelper = DatabaseHelper(this) και val db = dbHelper. Writable Database, καλείται η βάση δεδομένων και δίνει εντολή ώστε να μπορεί να καταχωρίσει στοιχεία. Μετά με την εντολή put(DatabaseHelper.TotalExpensesEntry. COLUMN_NAME_GENERAL_EXPENSES, totalExpensesInput), καταχωρούνται τα στοιχεία στον κατάλληλο πίνακα και στήλη.

Περιγραφή Layout

Με την λέξη layout αναφερόμαστε σε xml αρχεία που περιγράφουν τη διάταξη και τη δομή της οθόνης UI της εφαρμογής. Υπάρχουν διάφοροι τύποι layout όπως τα Linear Layout, τα Relative Layout, τα Constraint Layout και άλλα πολλά. Για την υλοποίηση της εφαρμογής έχουμε χρησιμοποιήσει τρία είδη από τα layout: τα Constraint Layout, τα Frame Layout και τα Linear Layout. Όπως βλέπουμε παρακάτω τα έχουμε χωρίσει σε 3 δικές μας κατηγορίες για την διευκόλυνση μας στην σύνδεση τους με τα αρχεία Kotlin.



Εικόνα 36 Layouts

Το Constraint Layout είναι ένα δυναμικό layout manager για το android studio το οποίο δημιουργήθηκε από την Google. Δημιουργήθηκε με σκοπό να βελτιώσει και να βοηθήσει τους χρήστες με τον τρόπο που τοποθετούνται τα στοιχεία διεπαφής (buttons, text, …) στην οθόνη. Επιτρέπει δηλαδή στους προγραμματιστές την δημιουργία πολύπλοκων και ευέλικτων διεπαφών χρήστη με βάση τους περιορισμούς που ορίζονται μεταξύ των στοιχείων. Αυτοί οι περιορισμοί καθορίζουν

τη θέση, το μέγεθος και τη σχετική τοποθέτηση των στοιχείων στην οθόνη χωρίς να χρειάζεται να τα προσδιορίσουμε με ακρίβεια. Αυτό καθιστά το Constraint Layout πολύ πιο ευέλικτο και ευανάγνωστο, ενώ παράλληλα βοηθάει στη δημιουργία διεπαφών που προσαρμόζονται αυτόματα σε διαφορετικές διαστάσεις οθόνων.

Βασικά χαρακτηριστικά ενός Constraint Layout είναι ο ορισμός περιορισμών μεταξύ των στοιχείων για τη θέση και το μέγεθος τους. Τα εργαλεία διεπαφής που έχει ενσωματωμένα, επιτρέπουν την εύκολη δημιουργία και επεξεργασία περιορισμών και δυναμική προσαρμοστικότητα των στοιχείων στις διαστάσεις των οθονών.

Το Frame Layout είναι ένα απλό layout manager στο android studio που διατηρεί τα στοιχεία του σε στοίβα. Σε αντίθεση με το ConstraintLayout, το FrameLayout δεν χρησιμοποιεί περιορισμούς για την τοποθέτηση των στοιχείων. Αντίθετα, κάθε στοιχείο είναι τοποθετημένο επάνω από το προηγούμενο, και το τελευταίο στοιχείο που προστίθεται στο FrameLayout είναι τοποθετημένο επάνω από όλα τα άλλα. Το FrameLayout είναι χρήσιμο όταν θέλουμε να εμφανίσουμε ένα μόνο στοιχείο τη φορά, όπως ένα μόνο κουμπί ή μια μόνο εικόνα, χωρίς να μας ενδιαφέρει η ακριβής τοποθέτηση του στοιχείου στην οθόνη.

Το Linear Layout είναι ένα από τα βασικά layout του Android Studio που τα στοιχεία του τοποθετούνται στοιχισμένα σε γραμμή ή σε στήλη. Δηλαδή είναι οριζόντιο ή κάθετο αναλόγως πως το έχουμε ορίσει εμείς. Τα στοιχεία του τοποθετούνται με τη σειρά εκτός αν τα έχουμε ορίσει εμείς με άλλους περιορισμούς.

Μέσα σε αυτά τα Layout χρησιμοποιούμε για την υλοποίηση της εφαρμογής διάφορα βασικά γραφικά στοιχεία όπως text view, edit text, recycler view, button, image view και horizontal scroll view.

Text View : Είναι ένα γραφικό στοιχείο του Android Studio που χρησιμοποιείται για την εμφάνιση κειμένου στην οθόνη του χρήστη. Το text view μπορεί να είναι μόνο για ανάγνωση ή και για τροποποίηση ανάλογα πως έχει ρυθμιστεί.

Edit Text : Είναι ένα γραφικό στοιχείο του Android Studio που χρησιμοποιείται για την εισαγωγή και την τροποποίηση κειμένου στην οθόνη του χρήστη.

Recycler View : Είναι ένα από τα πιο ισχυρά και ευέλικτα γραφικά στοιχεία του Android Studio που χρησιμοποιείται για την εμφάνιση μεγάλων συνόλων δεδομένων μέσα σε μία λίστα. Η βασική λειτουργία του είναι να επαναχρησιμοποιεί τα στοιχεία της λίστας που δεν είναι ορατά για την βελτίωση της απόδοσης και της μνήμης.

Button: Είναι ένα γραφικό στοιχείο του Android Studio που χρησιμοποιείται για την εκκίνηση μίας διεργασίας όταν πατηθεί από τον χρήστη.

image view: Είναι ένα γραφικό στοιχείο του Android Studio που χρησιμοποιείται για την εμφάνιση μίας εικόνας στην οθόνη. Μπορεί να είναι στατικό ή δυναμικό, δηλαδή να αλλάζει εικόνα με βάση την ενέργεια του χρήστη ή άλλες παραμέτρους.

horizontal scroll view : Είναι ένα γραφικό στοιχείο του Android Studio που επιτρέπει στον χρήστη να κάνει κύλιση οριζόντια για να δει τα στοιχεία που δεν φαίνονται στην οθόνη.

Μέσα από το Android Studio έχουμε δύο οπτικές των layout. Το code view και το UI view. Στο code view μπορούμε να προγραμματίσουμε με την χρήση της xml το γραφικό κομμάτι της εφαρμογής, ενώ στο UI view χρησιμοποιούμε την τεχνική drag and drop.

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:androi
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/background"
    tools:context=".Start">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:fontFamily="@font/pacifico"
        android:outlineProvider="none"
        android:shadowColor="#000000"
        android:shadowDx="2"
        android:shadowDy="2"
        android:shadowRadius="0.1"
        android:singleLine="false"
        android:text="Maintenance Fees"
        android:textAlignment="center"
        android:textColor="@color/orange"
        android:textSize="24sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.497"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.054" />
```

Εικόνα 37 Layout.xml

Μέσα στο layout.xml μπορούμε να γράψουμε κώδικα σε γλώσσα XML όπως βλέπουμε στο παραπάνω παράδειγμα. Στην αρχή δηλώνουμε το xml και το encoding που είναι απαραίτητα για τη σωστή λειτουργία του layout και στη συνέχεια ορίζουμε το Constraint Layout και τις απαραίτητες βιβλιοθήκες.

Στη συνέχεια δημιουργούμε ένα Text View με id textview. Με την εντολή android:layout_width="wrap_content" και android:layout_height="wrap_content" καθορίζουμε τον ορισμό του ύψους και του πλάτους ως αυτόματη προσαρμογή με βάση το περιεχόμενο του. Ορίζουμε την γραμματοσειρά και απενεργοποιούμε το περίγραμμα του κειμένου. Στη συνέχεια στις γραμμές 16 έως 19 ορίζουμε σκιά στο κείμενο. Με την εντολή android:singleLine="false", δηλώνεται ότι το κείμενο μπορεί

να αποτελείται από πολλαπλές γραμμές. Από την γραμμή 21 έως την γραμμή 24 δηλώνουμε το κείμενο με την στοίχιση, το χρώμα και τη γραμματοσειρά του. Τέλος από την γραμμή 25 μέχρι την γραμμή 30 ορίζουμε τα constraint layout.

Εικόνα 38 Layout UI View

Περιγραφή βάσης δεδομένων της εφαρμογής

Η βάση δεδομένων της εφαρμογής Maintenance Fees Calculator έχει δημιουργηθεί με SQLite. Ο κύριος λόγος που χρησιμοποιήθηκε αυτή η βάση δεδομένων είναι η αξιοπιστία της μαζί με την αυτονομία και την γρήγορη ανταπόκριση της. Εξίσου σημαντικό ρόλο έπαιξε το ότι αποθηκεύεται τοπικά και δεν χρειάζεται σύνδεση στο ίντερνετ.

Για την υλοποίηση της εφαρμογής έχουμε φτιάξει μία βάση δεδομένων με τέσσερις πίνακες, όπως βλέπουμε παρακάτω.

```
override fun onCreate(db: SQLiteDatabase) {
    db.execSQL(SQL_CREATE_USER_ENTRIES)
    db.execSQL(SQL_CREATE_ADDRESS_ENTRIES)
    db.execSQL(SQL_CREATE_APARTMENTS_ENTRIES)
    db.execSQL(SQL_CREATE_TOTAL_PAYMENTS_ENTRIES)
    db.execSQL(SQL_CREATE_TOTAL_EXPENSES_ENTRIES)
}
```

Εικόνα 39 Πίνακες Βάσης Δεδομένων

Σε καθένα από αυτούς τους πίνακες αποθηκεύονται τα ανάλογα στοιχεία όπως θα αναφερθούν παρακάτω.

```
private const val SQL_CREATE_USER_ENTRIES =
    "CREATE TABLE ${UsernameEntry.TABLE_NAME} (" +
            "${BaseColumns._ID} INTEGER PRIMARY KEY," +
            "${UsernameEntry.COLUMN_NAME_USERNAME} TEXT)"
```

Εικόνα 40 Πίνακας Username

Εδώ βλέπουμε την δημιουργία του πίνακα user entries που ανήκει στη βάση δεδομένων μας. Με την εντολή private const val SQL_CREATE_ADDRESS_ENTRIES

δηλώνουμε τον πίνακα και στη συνέχεια τον δημιουργούμε με την εντολή "CREATE TABLE ${UsernameEntry.TABLE_NAME}. Με την επόμενη εντολή, "${BaseColumns._ID} INTEGER PRIMARY KEY, δηλώνουμε ένα ID σαν primary key που θα μας βοηθήσει στην σύνδεση μεταξύ των πινάκων. Η τελευταία εντολή, "${UsernameEntry.COLUMN_NAME_USERNAME} TEXT)", δημιουργεί μία στήλη στον πίνακα που θα καταχωρείται το username του χρήστη και δηλώνεται σαν τύπου text, δηλαδή κείμενο.

```
private const val SQL_DELETE_USER_ENTRIES =
    "DROP TABLE IF EXISTS ${UsernameEntry.TABLE_NAME}"
```

Εικόνα 41 Διαγραφή πίνακα Username

Με αυτή την εντολή διαγράφουμε τον πίνακα που έχουμε δημιουργήσει προηγουμένως.

```
private const val SQL_CREATE_ADDRESS_ENTRIES =
    "CREATE TABLE ${AddressEntry.TABLE_NAME} (" +
            "${BaseColumns._ID} INTEGER PRIMARY KEY," +
            "${AddressEntry.COLUMN_NAME_ADDRESS} TEXT," +
            "${AddressEntry.COLUMN_NAME_ADDRESS_NUMBER} TEXT," +
            "${AddressEntry.COLUMN_NAME_FLOORS} TEXT," +
            "${AddressEntry.COLUMN_NAME_FLATS} TEXT," +
            "${AddressEntry.COLUMN_NAME_TOTAL_SQ_METERS} FLOAT," +
            "${AddressEntry.COLUMN_NAME_ELEVATOR_CHECKED} INTEGER," +
            "${AddressEntry.COLUMN_NAME_HEATING_CHECKED} INTEGER," +
            "${AddressEntry.COLUMN_NAME_M2CALC_CHECKED} INTEGER)"

private const val SQL_DELETE_ADDRESS_ENTRIES =
    "DROP TABLE IF EXISTS ${AddressEntry.TABLE_NAME}"
```

Εικόνα 42 Πίνακας Address

Εδώ βλέπουμε την δημιουργία του δεύτερου πίνακα της βάσης μας με όνομα Address Entry. Πάλι έχουμε δηλώσει σαν primary key ένα ID και στη συνέχεια

δημιουργούμε και δηλώνουμε τις υπόλοιπες στήλες που θα χρειαστούμε. Οι στήλες αυτές είναι η address, η address number, η floors, η flats, η total sq meters, η elevator checked, η heating checked και η m2calc checked. Με το τύπο float δηλώνουμε αριθμό με δεκαδικά ψηφία και με τον τύπο integer δηλώνουμε αριθμούς χωρίς δεκαδικά ψηφία. Στη συνέχει βάζουμε μία εντολή για την διαγραφή αυτού του πίνακα που έχουμε δημιουργήσει.

```
private const val SQL_CREATE_APARTMENTS_ENTRIES =
    "CREATE TABLE ${ApartmentsEntry.TABLE_NAME} (" +
            "${BaseColumns._ID} INTEGER PRIMARY KEY," +
            "${ApartmentsEntry.COLUMN_NAME_NAME} TEXT," +
            "${ApartmentsEntry.COLUMN_NAME_SURNAME} TEXT," +
            "${ApartmentsEntry.COLUMN_NAME_FLOOR} TEXT," +
            "${ApartmentsEntry.COLUMN_NAME_PHONE} TEXT," +
            "${ApartmentsEntry.COLUMN_NAME_EMAIL} TEXT," +
            "${ApartmentsEntry.COLUMN_NAME_SQ_METERS} FLOAT," +
            "${ApartmentsEntry.COLUMN_NAME_SHARED_FEES} FLOAT," +
            "${ApartmentsEntry.COLUMN_NAME_ELEVATOR_FEES} FLOAT," +
            "${ApartmentsEntry.COLUMN_NAME_HEATING_UNITS} FLOAT," +
            "${ApartmentsEntry.COLUMN_NAME_BUILDING_ID} INTEGER," +

            "FOREIGN KEY(${ApartmentsEntry.COLUMN_NAME_BUILDING_ID}) " +
            "REFERENCES ${AddressEntry.TABLE_NAME}" +
            "(${BaseColumns._ID}) ON DELETE CASCADE ON UPDATE CASCADE)"

private const val SQL_DELETE_APARTMENTS_ENTRIES =
    "DROP TABLE IF EXISTS ${ApartmentsEntry.TABLE_NAME}"
```

Εικόνα 43 Πίνακας Apartments

Σε αυτόν τον πίνακα με την ίδια λογική τον κατασκευάζουμε και τον δηλώνουμε προσθέτοντας τις στήλες id που δηλώνεται πάλι σαν primary key και τις υπόλοιπες για τα στοιχεία που θα χρειαστεί να αποθηκεύσουμε για τους ενοίκους. Αυτές είναι η name, η surname, η floor, η phone, η email, η sqm meters, η shared feed, η elevator fees, η heating units και η building id. Εδώ έχουμε και μία επιπλέον καταχώρηση το "FOREIGN KEY(${ApartmentsEntry.COLUMN_NAME_BUILDING_ID}) REFERENCES ${AddressEntry.TABLE_NAME}(${BaseColumns._ID}) ON DELETE CASCADE ON UPDATE CASCADE)". Με αυτή την εντολή δημιουργούμε την σύνδεση

των δύο πινάκων, του apartments με τον address. Στη συνέχεια, παρουσιάζουμε και τους άλλους δύο πίνακες που έχουν κατασκευαστεί και δηλωθεί με την ίδια λογική.

```
private const val SQL_CREATE_TOTAL_PAYMENTS_ENTRIES =
    "CREATE TABLE ${TotalPaymentsEntry.TABLE_NAME} (" +
            "${BaseColumns._ID} INTEGER PRIMARY KEY," +
            "${TotalPaymentsEntry.COLUMN_NAME_NAME} TEXT," +
            "${TotalPaymentsEntry.COLUMN_NAME_SURNAME} TEXT," +
            "${TotalPaymentsEntry.COLUMN_NAME_PHONE} NUMBER," +
            "${TotalPaymentsEntry.COLUMN_NAME_MONTH_YEAR} FLOAT," +
            "${TotalPaymentsEntry.COLUMN_NAME_TOTAL_FEES} FLOAT," +
            "${TotalPaymentsEntry.COLUMN_NAME_GENERAL_PAY_FEES} FLOAT," +
            "${TotalPaymentsEntry.COLUMN_NAME_ELEVATOR_PAY_FEES} FLOAT," +
            "${TotalPaymentsEntry.COLUMN_NAME_HEATING_PAY_FEES} FLOAT," +
            "${TotalPaymentsEntry.COLUMN_NAME_APARTMENTS_ID} INTEGER," +


            "FOREIGN KEY(${TotalPaymentsEntry.COLUMN_NAME_APARTMENTS_ID}) REFERENCES " +
            "${ApartmentsEntry.TABLE_NAME}(${BaseColumns._ID}) " +
            "ON DELETE CASCADE ON UPDATE CASCADE)"

private const val SQL_DELETE_TOTAL_PAYMENTS_ENTRIES =
    "DROP TABLE IF EXISTS ${TotalPaymentsEntry.TABLE_NAME}"
```

Εικόνα 44 Πίνακας Payments

```
private const val SQL_CREATE_TOTAL_EXPENSES_ENTRIES =
    "CREATE TABLE ${TotalExpensesEntry.TABLE_NAME} (" +
            "${BaseColumns._ID} INTEGER PRIMARY KEY," +
            "${TotalExpensesEntry.COLUMN_NAME_ADDRESS} FLOAT," +
            "${TotalExpensesEntry.COLUMN_NAME_MONTH_YEAR} FLOAT," +
            "${TotalExpensesEntry.COLUMN_NAME_TOTAL_EXPENSES} FLOAT," +
            "${TotalExpensesEntry.COLUMN_NAME_GENERAL_EXPENSES} FLOAT," +
            "${TotalExpensesEntry.COLUMN_NAME_ELEVATOR_EXPENSES} FLOAT," +
            "${TotalExpensesEntry.COLUMN_NAME_HEATING_EXPENSES} FLOAT," +
            "${TotalExpensesEntry.COLUMN_NAME_BUILDING_ID} INTEGER," +


            "FOREIGN KEY(${TotalExpensesEntry.COLUMN_NAME_BUILDING_ID}) " +
            "REFERENCES ${AddressEntry.TABLE_NAME}(${BaseColumns._ID}) " +
            "ON DELETE CASCADE ON UPDATE CASCADE)"

private const val SQL_DELETE_TOTAL_EXPENSES_ENTRIES =
    "DROP TABLE IF EXISTS ${TotalExpensesEntry.TABLE_NAME}"
```

Εικόνα 45 Πίνακας total expenses

Στη συνέχεια με την παράθεση μερικών εικόνων, σας παρουσιάζουμε τον ορισμό των στηλών για τον κάθε πίνακα που έχουμε δημιουργήσει.

```
object UsernameEntry : BaseColumns {
    const val TABLE_NAME = "username"
    const val COLUMN_NAME_USERNAME = "username"
}
```

Εικόνα 46 Ορισμός πίνακα username

```
object AddressEntry : BaseColumns {
    const val TABLE_NAME = "address"
    const val COLUMN_NAME_ADDRESS = "address"
    const val COLUMN_NAME_ADDRESS_NUMBER = "address_number"
    const val COLUMN_NAME_FLOORS = "floors"
    const val COLUMN_NAME_FLATS = "flats"
    const val COLUMN_NAME_TOTAL_SQ_METERS = "total_sq_meters"
    const val COLUMN_NAME_ELEVATOR_CHECKED = "elevator"
    const val COLUMN_NAME_HEATING_CHECKED = "heating"
    const val COLUMN_NAME_M2CALC_CHECKED = "m2calc"
}
```

Εικόνα 47 Ορισμός πίνακα address

```
object ApartmentsEntry : BaseColumns {
    const val TABLE_NAME = "apartment"
    const val COLUMN_NAME_NAME = "name"
    const val COLUMN_NAME_SURNAME = "surname"
    const val COLUMN_NAME_FLOOR = "floor"
    const val COLUMN_NAME_PHONE = "phone"
    const val COLUMN_NAME_EMAIL = "email"
    const val COLUMN_NAME_SQ_METERS = "sq_meters"
    const val COLUMN_NAME_SHARED_FEES = "shared_fees"
    const val COLUMN_NAME_ELEVATOR_FEES = "elevator_fees"
    const val COLUMN_NAME_HEATING_UNITS = "heating_units"
    const val COLUMN_NAME_BUILDING_ID = "building_id"
}
```

Εικόνα 48 Ορισμός πίνακα apartment

```
object TotalPaymentsEntry : BaseColumns {
    const val TABLE_NAME = "totalPayments"
    const val COLUMN_NAME_NAME = "name"
    const val COLUMN_NAME_SURNAME = "surname"
    const val COLUMN_NAME_PHONE = "phone"
    const val COLUMN_NAME_MONTH_YEAR = "yearMonth"
    const val COLUMN_NAME_TOTAL_FEES = "total_fees"
    const val COLUMN_NAME_GENERAL_PAY_FEES = "general_pay_fees"
    const val COLUMN_NAME_ELEVATOR_PAY_FEES = "elevator_pay_fees"
    const val COLUMN_NAME_HEATING_PAY_FEES= "heating_pay_fees"

    const val COLUMN_NAME_APARTMENTS_ID = "apartment_id"
}
```

Εικόνα 49 Ορισμός πίνακα total payments

```
object TotalExpensesEntry : BaseColumns {
    const val TABLE_NAME = "totalExpenses"
    const val COLUMN_NAME_ADDRESS = "address"
    const val COLUMN_NAME_MONTH_YEAR = "yearMonth"
    const val COLUMN_NAME_TOTAL_EXPENSES = "total_expenses"
    const val COLUMN_NAME_GENERAL_EXPENSES = "general_expenses"
    const val COLUMN_NAME_ELEVATOR_EXPENSES = "elevator_expenses"
    const val COLUMN_NAME_HEATING_EXPENSES = "heating_expenses"
    const val COLUMN_NAME_BUILDING_ID = "building_id"

}
```

Εικόνα 50 Ορισμός πίνακα total expenses

Έπειτα, με τις κατάλληλες function αποθηκεύουμε τα διάφορα δεδομένα που έχει καταχωρήσει ο χρήστης, στα πεδία που αντιστοιχούν μέσα στη βάση δεδομένων που έχουμε δημιουργήσει. Επιπροσθέτως, ρησιμοποιούμε function και για τις πράξεις μεταξύ στηλών και πινάκων που χρειάστηκαν για τη σωστή λειτουργία της εφαρμογής, όπως φαίνεται από τις παρακάτω εικόνες.

```
fun updateApartment(apartment: Apartment): Int {
    val db = writableDatabase

    val values = ContentValues().apply { this: ContentValues
        put(ApartmentsEntry.COLUMN_NAME_NAME, apartment.name)
        put(ApartmentsEntry.COLUMN_NAME_SURNAME, apartment.surname)
        put(ApartmentsEntry.COLUMN_NAME_FLOOR, apartment.floor)
        put(ApartmentsEntry.COLUMN_NAME_PHONE, apartment.phone)
        put(ApartmentsEntry.COLUMN_NAME_EMAIL, apartment.email)
        put(ApartmentsEntry.COLUMN_NAME_SQ_METERS, apartment.sqMeters)
        put(ApartmentsEntry.COLUMN_NAME_SHARED_FEES, apartment.sharedFees)
        put(ApartmentsEntry.COLUMN_NAME_ELEVATOR_FEES, apartment.elevatorFees)
    }

    val selection = "${BaseColumns._ID} = ?"
    val selectionArgs = arrayOf(apartment.id.toString())

    val rowsAffected = db.update(ApartmentsEntry.TABLE_NAME, values, selection, selectionArgs)

    db.close()

    return rowsAffected
}
```

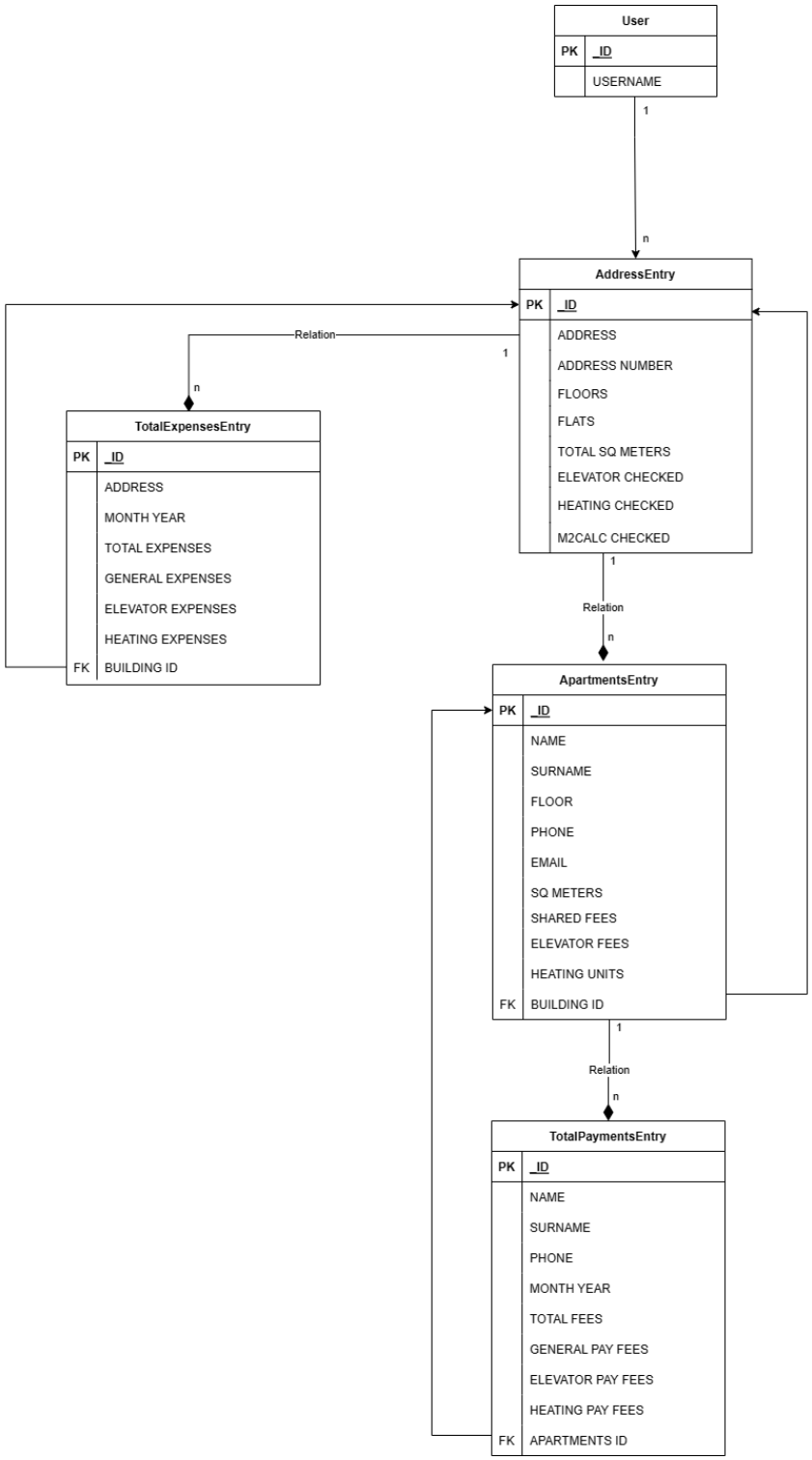
Εικόνα 51 Καταχώριση στοιχείων διαμερίσματος

```
fun copyAddress(buildingId: Long) {
    val db = writableDatabase

    val query = """
    UPDATE ${TotalExpensesEntry.TABLE_NAME}
    SET
        ${TotalExpensesEntry.COLUMN_NAME_ADDRESS} = (
            SELECT ${AddressEntry.COLUMN_NAME_ADDRESS}
            FROM ${AddressEntry.TABLE_NAME}
            WHERE ${AddressEntry.TABLE_NAME}.${BaseColumns._ID} = $buildingId
        ),
        ${TotalExpensesEntry.COLUMN_NAME_TOTAL_EXPENSES} =
            COALESCE(${TotalExpensesEntry.COLUMN_NAME_GENERAL_EXPENSES}, 0) +
            COALESCE(${TotalExpensesEntry.COLUMN_NAME_ELEVATOR_EXPENSES}, 0) +
            COALESCE(${TotalExpensesEntry.COLUMN_NAME_HEATING_EXPENSES}, 0)
    WHERE ${TotalExpensesEntry.COLUMN_NAME_BUILDING_ID} = $buildingId
""".trimIndent()
```

Εικόνα 52 Πράξεις στηλών

Διαγράμματα ERD

Τα διαγράμματα Entity Relationship Diagrams (ERD) αναφέρονται σε διαγράμματα που χρησιμοποιούνται στη βάση δεδομένων για την απεικόνιση των οντοτήτων, τη σχέση μεταξύ τους και τα χαρακτηριστικά τους. Με λίγα λόγια μας βοηθούν στην πιο εύκολη κατανόηση της βάσης δεδομένων. Τα βασικά στοιχεία των Entity Relationship Diagrams μπορούμε να τα διαχωρίσουμε σε τρεις κατηγορίες: τις σχέσεις, τις οντότητες και τα χαρακτηριστικά τους. Με τις οντότητες αναφερόμαστε στα στοιχεία του πίνακα που θέλουμε να απεικονίσουμε στη βάση δεδομένων, για παράδειγμα ένας πίνακας με τους ένοικους της πολυκατοικίας. Με τον όρο σχέσεις ορίζουμε πως συνδέονται οι οντότητες μεταξύ τους, δηλαδή μία πολυκατοικία μπορεί να έχει πολλά διαμερίσματα. Με τον όρο χαρακτηριστικά αναφερόμαστε στα χαρακτηριστικά των οντοτήτων, για παράδειγμα ένας ένοικος μπορεί να έχει ένα όνομα και ένα τηλέφωνο. Με λίγα λόγια, τα Entity Relationship Diagrams χρησιμοποιούνται στα στάδια σχεδιασμού της βάσης δεδομένων, προκειμένου να μας προσφέρει μία οπτική αναπαράσταση της δομής της και των σχέσεων των δεδομένων που αποθηκεύονται σε αυτή. Αυτό βοηθάει τον προγραμματιστή στην υλοποίηση και στην κατανόηση των απαιτήσεων της.



Εικόνα 53 ERD Διάγραμμα Βάσης Δεδομένων

Με το παραπάνω διάγραμμα Entity Relationship Diagrams (ERD) παρουσιάζουμε την δομή της βάσης δεδομένων που έχει δημιουργηθεί για τις ανάγκες της εφαρμογής. Για την διευκόλυνση της ανάγνωσης της εικόνας θα

επεξηγηθούν μερικά από τα σύμβολα των ERD διαγραμμάτων. Ο συμβολισμός ένα προς πολλά (1→n) αναφαίρεται στη σχέση δύο οντοτήτων στη βάση δεδομένων. Το ένα σημαίνει ότι μία εγγραφή στη πρώτη οντότητα σχετίζεται με μόνο μία εγγραφή στη δεύτερη οντότητα και το n ότι μία εγγραφή στη δεύτερη οντότητα σχετίζεται με πολλές εγγραφές στη πρώτη οντότητα. Ενώ ο συμβολισμός πολλά προς ένα (n→1) σημαίνει μία εγγραφή στη πρώτη οντότητα σχετίζεται με πολλές εγγραφές στη δεύτερη οντότητα και μία εγγραφή στη δεύτερη οντότητα σχετίζεται μόνο με μία εγγραφή στη πρώτη οντότητα.

Στη παραπάνω εικόνα 51 βλέπουμε ότι ένας user μπορεί να έχει πολλά Address Entry δηλαδή πολλά κτήρια, αλλά ένα κτήριο μπορεί να έχει έναν user. Το ίδιο συμβαίνει και με τους υπόλοιπους πίνακες. Ένα Address Entry μπορεί να έχει πολλά Total Expenses Entry και πολλά Apartments Entry, αλλά τα Total Expenses Entry και τα Apartments Entry μπορούν να έχουν μόνο ένα address Entry. Ένα Apartments Entry μπορεί να έχει πολλά Total Payments Entry αλλά ένα Total Payments Entry μπορεί να έχει μόνο ένα Apartments Entry.

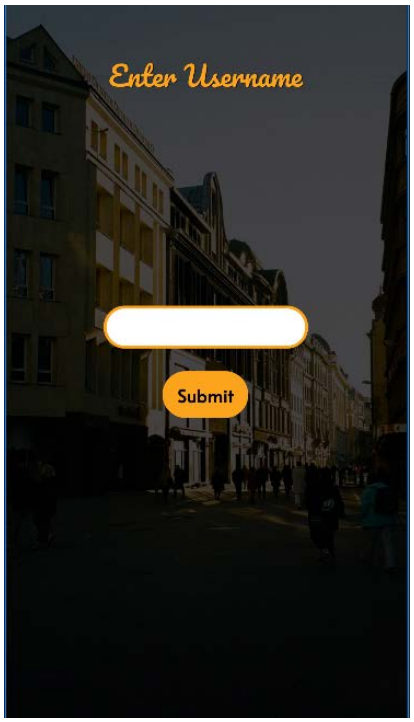
Συνοπτικά μπορούμε να πούμε ότι κάθε Apartment ανήκει σε ένα Address, κάθε Total Payment ανήκει σε ένα Apartment και κάθε Total Expenses ανήκει σε ένα Address άρα σε κάθε σύνδεση έχουμε πολλά προς ένα relationship.

Περιγραφή εκτέλεσης της εφαρμογής

Ανοίγοντας την εφαρμογή ο χρήστης αντικρίζει μια αρχική οθόνη με το λογότυπο της εφαρμογής και ένα κουμπί Start για την εκκίνηση της διαδικασίας καταχώρισης.

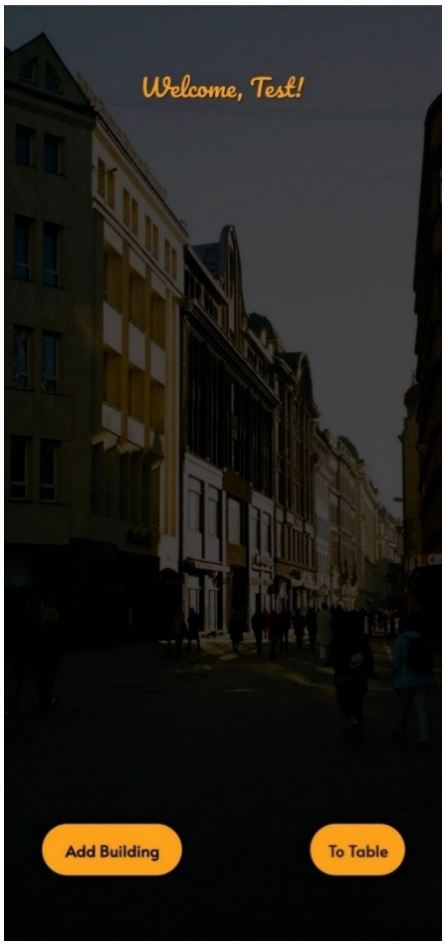
Εικόνα 54 Start Screen

Πατώντας το κουμπί αλλάζει καρτέλα στην οθόνη και αναγράφουμε το όνομα χρήστη που επιθυμούμε.



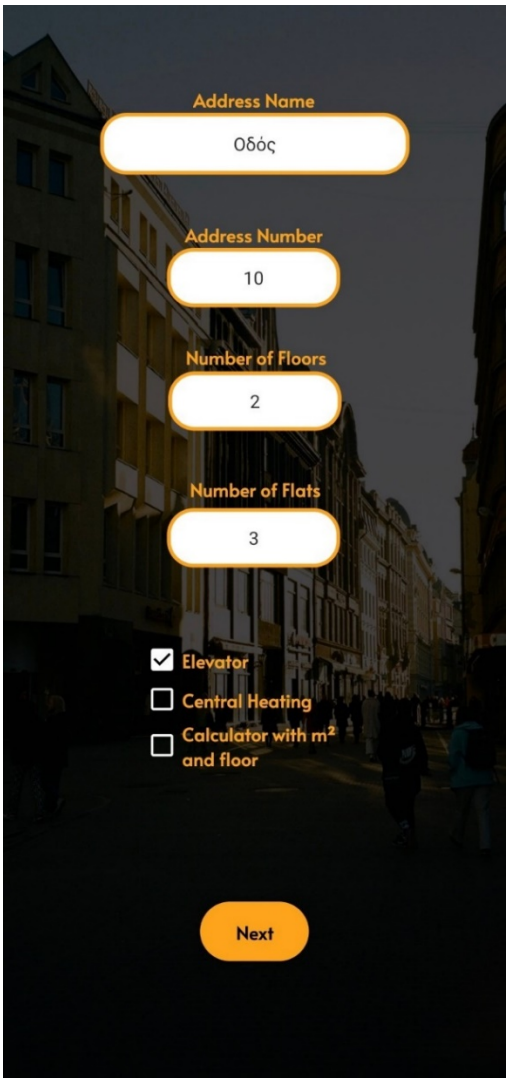
Εικόνα 55 Όνομα χρήστη

Καταχωρώντας το όνομα χρήστη και πατώντας το κουμπί Submit, πηγαίνουμε στην επόμενη καρτέλα, στην οποία υπάρχουν δύο κουμπιά. Το πρώτο κουμπί είναι το Add Building, με το οποίο μπορούμε να καταχωρίσουμε την πολυκατοικία που θέλουμε και να την εμφανίσει στην οθόνη μέσα σε ένα recycler view. Το άλλο κουμπί είναι το To Table που μας επιτρέπει να δούμε τα κοινόχρηστα που έχουμε υπολογίσει παλαιότερα.



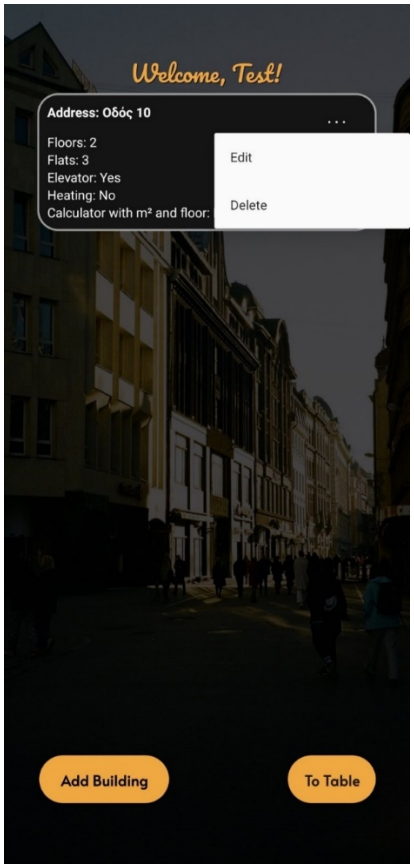
Εικόνα 56 Κτήρια

Πατώντας το Add Building ανοίγει μια καρτέλα στην οποία αναγράφουμε την οδό, τον αριθμό των ορόφων και των διαμερισμάτων. Επιπλέον, συμπληρώνουμε τα εν λόγω πεδία, αν το κτήριο διαθέτει ανελκυστήρα, κεντρική θέρμανση και αν θέλουμε να υπολογίσουμε τα κοινόχρηστα βάση τετραγωνικών μέτρων και ορόφων.



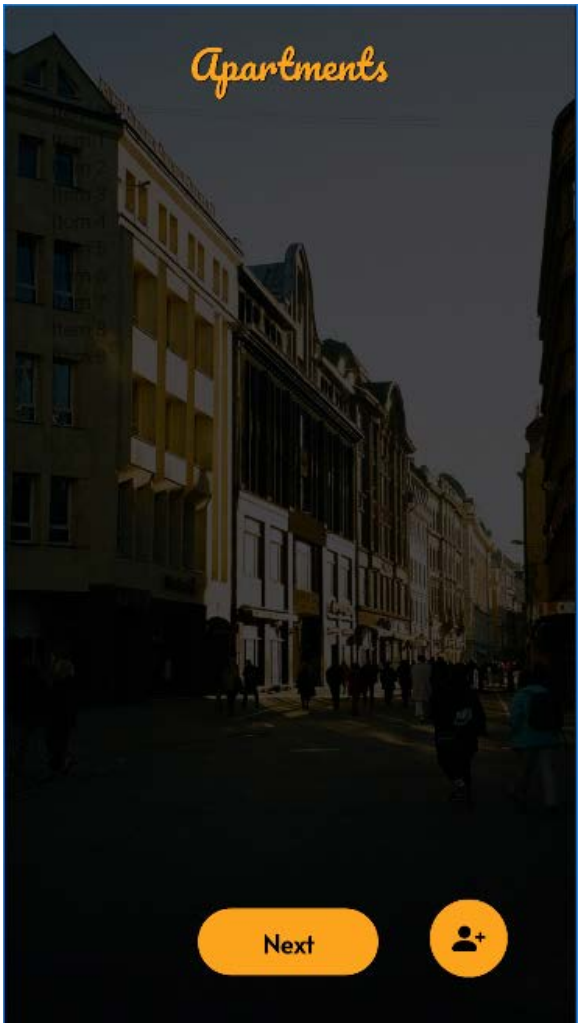
Εικόνα 57 Προσθήκη Κτηρίου

Μετά τη δημιουργία ενός κτηρίου από το μενού του (τρεις τελείες), μπορούμε να επεξεργαστούμε τα στοιχεία που έχουμε καταχωρίσει ή και να τα διαγράψουμε εντελώς.



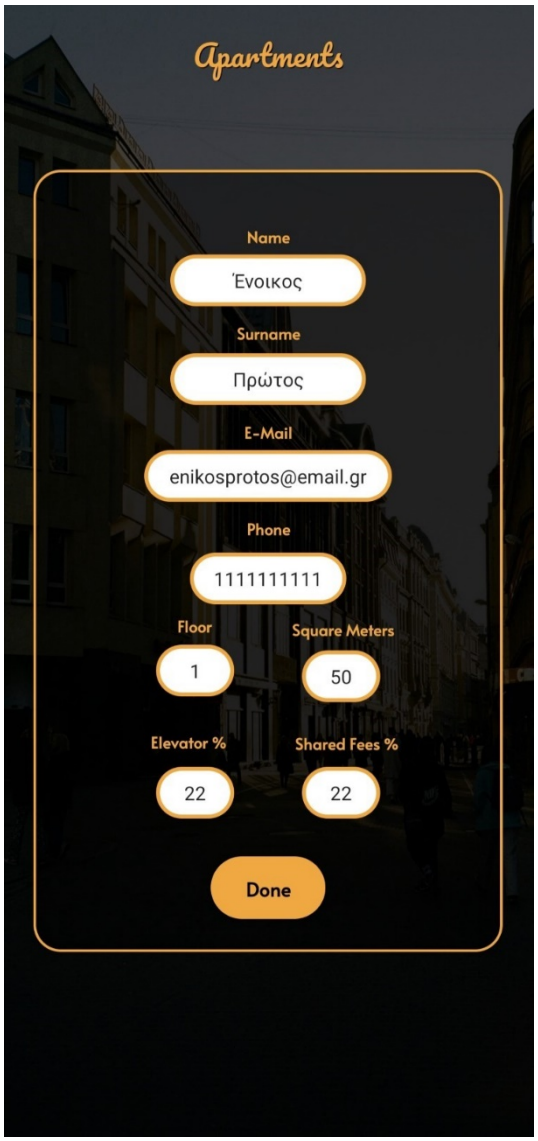
Εικόνα 58 Επεξεργασία / Διαγραφή Κτηρίου

Πατώντας πάνω σε κάποιο κτήριο που έχουμε δημιουργήσει πηγαίνουμε στην επόμενη καρτέλα, όπου προσθέτουμε τους ενοίκους του κτηρίου.



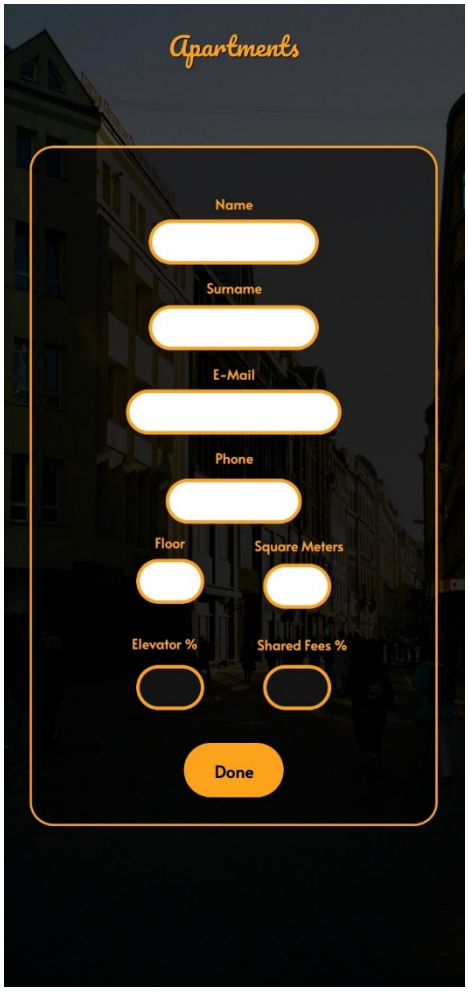
Εικόνα 59 Ένοικοι Κτηρίου

Πατώντας το fab button ανοίγει μία καρτέλα στην οποία θα πρέπει να συμπληρώσουμε τα στοιχεία του ενοίκου. Συμπληρώνουμε το όνομα, το επίθετο, προαιρετικά ένα email, το τηλέφωνο, τον όροφο και τα τετραγωνικά μέτρα του διαμερίσματος.



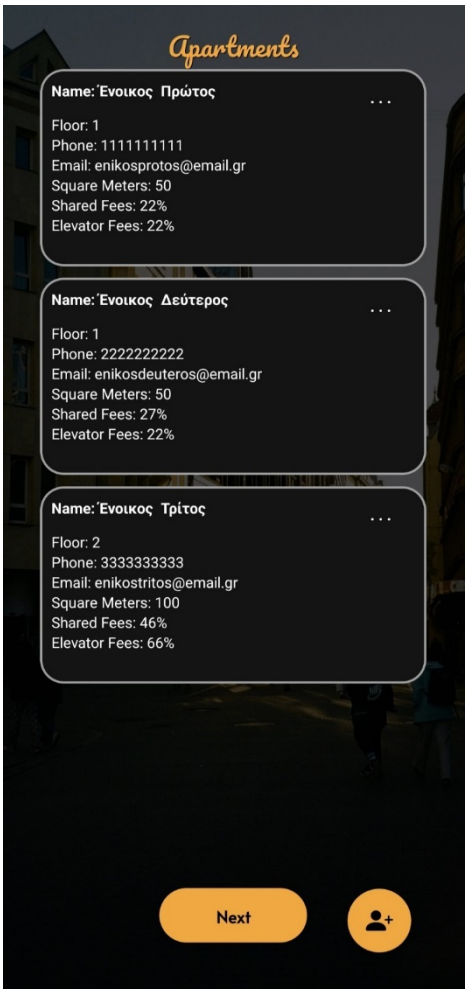
Εικόνα 60 Προσθήκη Ενοίκου

Ανάλογα τις επιλογές που έχουμε κάνει στην καρτέλα που προσθέσαμε το κτήριο, αν είναι ενεργά προσθέτουμε το ποσοστό χρέωσης για τον ανελκυστήρα και τα κοινόχρηστα.



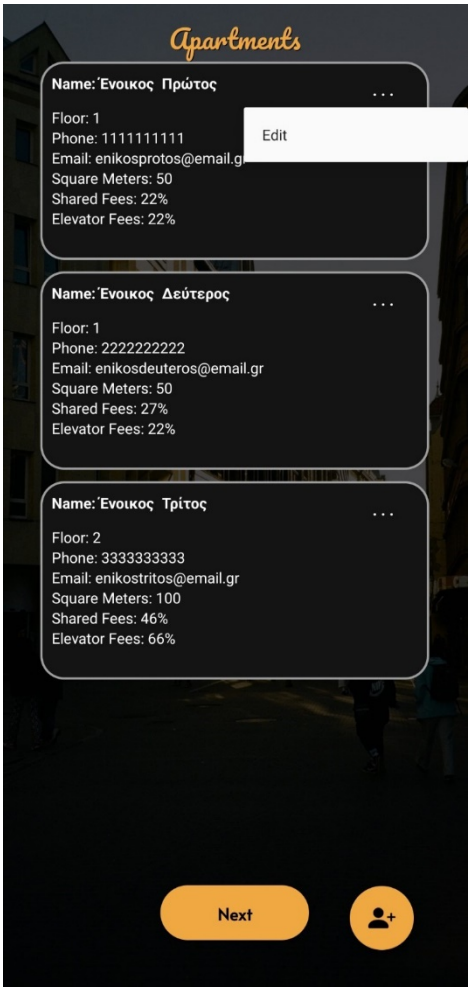
Εικόνα 61 Προσθήκη Ενοίκου

Πατώντας το κουμπί Done πηγαίνουμε ξανά στην καρτέλα με τους ενοίκους και μας εμφανίζει αυτούς που έχουμε προσθέσει μέσα σε ένα recycler view.



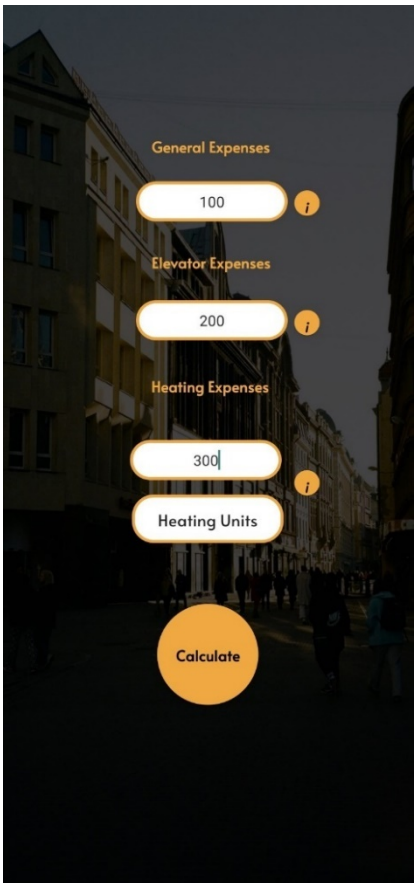
Εικόνα 62 Εμφάνιση Ενοίκων

Μετά τη δημιουργία του διαμερίσματος από το μενού του (τρεις τελείες), μπορούμε να επεξεργαστούμε τα στοιχεία για τυχόν αλλαγή τηλεφώνου, email ή και του ενοίκου.

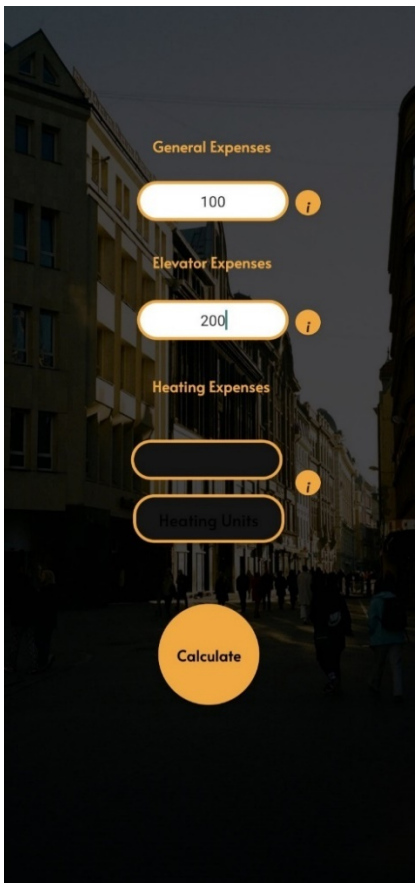


Εικόνα 63 Επεξεργασία Ενοίκου

Με το Next πάμε στο επόμενο βήμα. Εκεί έχουμε τρία edit text και δύο κουμπιά. Στο πρώτο text καταχωρούμε τα γενικά έξοδα της πολυκατοικίας όπως την καθαρίστρια, το κοινόχρηστο ρεύμα, το κοινόχρηστο νερό και ότι άλλο έξοδο προκύψει. Αν είναι ενεργοποιημένο ανάλογα τις προηγούμενες επιλογές μας, στο επόμενο text καταχωρούμε τα έξοδα για την συντήρηση του ανελκυστήρα ή για τυχόν αλλαγή του.



Εικόνα 64 Προσθήκη Εξόδων



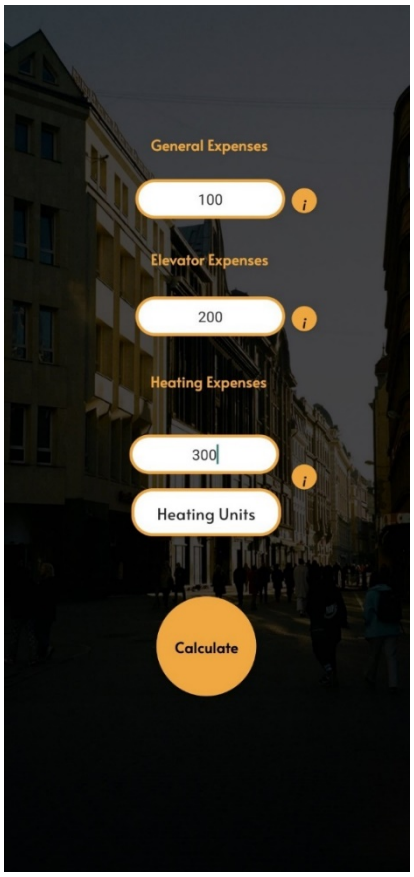
Εικόνα 65 Προσθήκη Εξόδων

Στη συνέχεια, αν υπάρχει κεντρική θέρμανση και είναι ενεργοποιημένο το πεδίο, στο επόμενο edit text και στο κουμπί αναγράφουμε το σύνολο που πληρώσαμε για την ανανέωση του καυσίμου και την συντήρηση του καυστήρα. Πατώντας το κουμπί Heating Units εμφανίζεται μία καρτέλα που αναγράφει το

ονοματεπώνυμο κάθε ένοικου και δίπλα καταχωρούμε τις μονάδες που κατανάλωσε το κάθε διαμέρισμα.

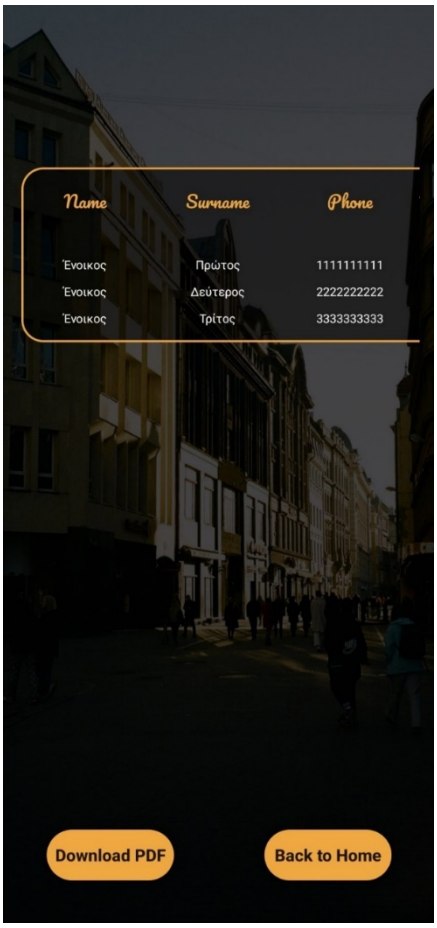
Εικόνα 66 Προσθήκη μονάδων θέρμανσης

Πατώντας το κουμπί OK επιστρέφουμε στην προηγούμενη καρτέλα και πατάμε το κουμπί Calculate.



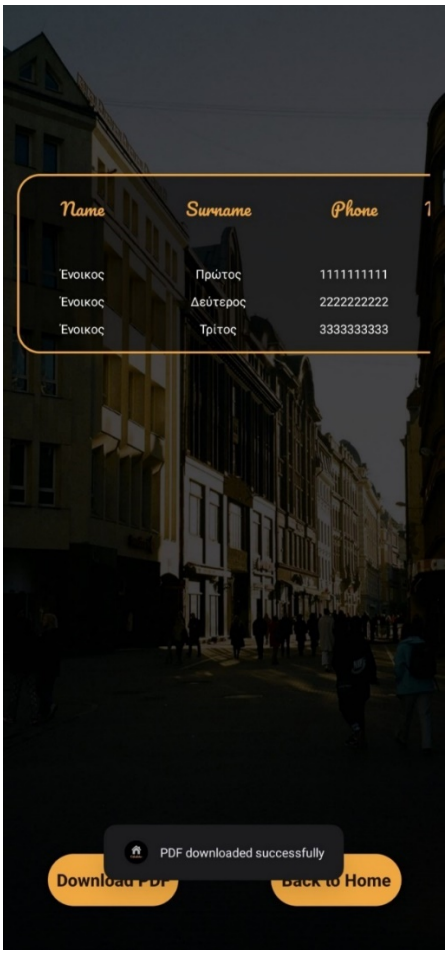
Εικόνα 67 Προσθήκη Εξόδων

Τέλος, μας εμφανίζει μία καρτέλα με ένα recycler view που αναγράφει τα στοιχεία του ένοικου, το ποσό της κάθε κατηγορίας και το συνολικό ποσό που πρέπει να πληρώσει. Στο κάτω μέρος της καρτέλας έχει δύο κουμπιά το Download PDF και το Back to Home.



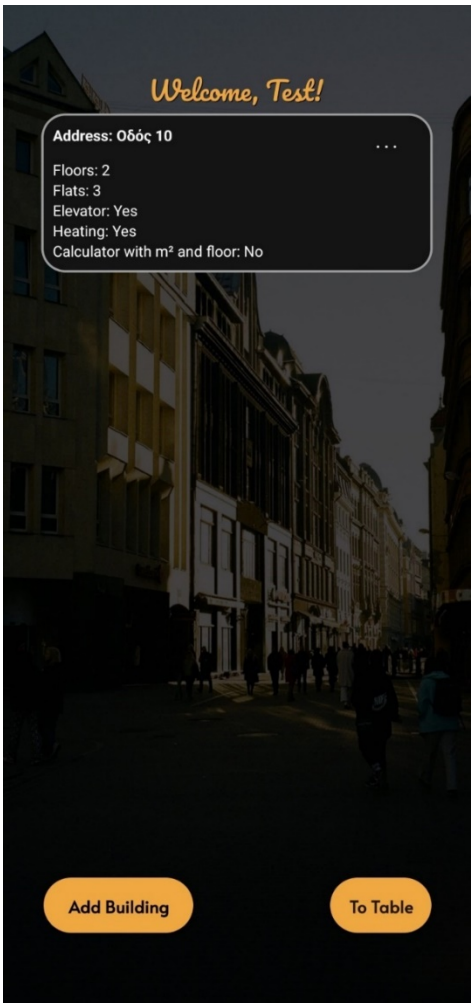
Εικόνα 68 Χρέωση Ενοίκου

Με το Download PDF κατεβάζει στη συσκευή του χρήστη ένα αρχείο pdf, στο οποίο αναγράφονται τα τελευταία κοινόχρηστα που έχουν υπολογιστεί. Ο χρήστης μπορεί να το βρει στα στοιχεία λήψης και να το εκτυπώσει ή να το στείλει στους ένοικους.



Εικόνα 69 Αποθήκευση PDF

Με το Back to Home πηγαίνουμε στην καρτέλα που εμφανίζονται οι καταχωρημένες πολυκατοικίες.



Εικόνα 70 Πολυκατοικίες

Συμπεράσματα και μελλοντικές επεκτάσεις

Κατά την υλοποίηση της εφαρμογής χρησιμοποιήθηκαν οι τελευταίες εκδόσεις του android studio, το οποίο εξελίσσεται καθημερινά. Σκοπός της εργασίας ήταν η μελέτη του android studio και η γλώσσα προγραμματισμού Kotlin. Προσωπικά, επιτεύχθηκε μεγάλη εξέλιξη μέσα από τον προγραμματισμό αυτής της εφαρμογής σε γνώσεις τόσο προγραμματιστικές όσο και ανάλυσης σκέψεων και ιδεών. Είναι μία εργασία που χρησιμοποιεί μεγάλο εύρος εργαλείων, τα οποία μπορούν να χρησιμοποιηθούν για τη δημιουργία διαφόρων ειδών εφαρμογών. Αυτός ήταν και ένας από τους βασικούς λόγους της επιλογής αυτού του θέματος. Η απόκτηση γνώσης για τα θεμέλια της δημιουργίας των μελλοντικών εφαρμογών.

Η εφαρμογή της πτυχιακής εργασίας περιέχει πολλά βασικά στοιχεία και δυνατότητες για την επίλυση του προβλήματος που συναντάμε στον υπολογισμό και την πληρωμή των κοινοχρήστων σε μία πολυκατοικία. Οι δυσκολίες που συναντήθηκαν κατά τον προγραμματισμό της εφαρμογής επιλύθηκαν άμεσα, αλλά παρόλα αυτά ήταν ανέφικτη η παροχή όλων ή παραπάνω λειτουργιών μέσα στο χρονικό όριο της πτυχιακής εργασίας.

Τα σημεία που θα μπορούσαν να βελτιωθούν στην εφαρμογή, είναι αρχικά η κάλυψη ενός ευρύτερου φάσματος γλωσσών, η προσθήκη βάση δεδομένων να γίνει σε SQL και να υποστηρίζεται διαδικτυακά από κάποιον Server. Αυτό θα βοηθούσε στη δημιουργία προφίλ διαχειριστή ή ένοικου και την σύνδεση τους με όνομα και κωδικό πρόσβασης. Με αυτό το τρόπο θα μπορούσε να γίνει η αποστολή και η πληρωμή των λογαριασμών μέσα από την εφαρμογή και ο διαχειριστής να ελέγχει αν έχει πληρωθεί ο λογαριασμός και από ποιον. Ακόμη, θα μπορούσε να προστεθεί ένα μενού για επιλογή μήνα και έτους πριν από την καταχώρηση εξόδων για την πιθανή διόρθωση ή την καταχώρηση παλαιότερων και μελλοντικών εξόδων. Θα μπορούσαμε να προσθέσουμε ακόμα περισσότερους περιορισμούς και

επισημάνσεις σε όλα τα πεδία που συμπληρώνει ο χρήστης. Αυτό θα βοηθήσει στην αποφυγή λανθασμένων καταχωρήσεων από τους χρήστες, με αποτέλεσμα να κάνουμε την εφαρμογή ακόμα πιο απλή στη χρήση της. Τέλος, η συνεχής παρακολούθηση των ενημερώσεων για την ομαλή λειτουργία της εφαρμογής, καθώς επίσης και την αξιολόγηση του feedback των χρηστών για περεταίρω βελτίωσης της.

Βιβλιογραφία

[1] Neil S., “Android Studio Iguana Essentials Payload Media” 2024

[2] Android Studio “Meet Android Studio” [13/11/2023] https://developer.android.com/studio/intro

[3] Android Studio “Database Inspector” [20/12/2023] https://developer.android.com/studio/inspect/database

[4] Android Studio “Releases” [03/03/2024] https://developer.android.com/studio/releases

[5] Android Studio “Layout Inspector” [18/11/2023] https://developer.android.com/develop/ui/compose/tooling/layout-inspector

[6] Android Studio “Logcat” [18/11/2023] https://developer.android.com/studio/debug/logcat

[7] Android Studio “Emulator” [15/11/2023] https://developer.android.com/studio/run/emulator

[8] Android Studio “Lifecycle” [22/03/2024] https://developer.android.com/jetpack/androidx/releases/lifecycle

[9] Android Studio “Core” [22/03/2024] https://developer.android.com/jetpack/androidx/releases/core

[10] Android Studio “Appcompat” [22/03/2024] https://developer.android.com/jetpack/androidx/releases/appcompat

[11] Android Studio “Material Design for Android” [23/03/2024] https://developer.android.com/develop/ui/views/theming/look-and-feel

[12] Android Studio “ConstraintLayout” [07/12/2023]

https://developer.android.com/reference/androidx/constraintlayout/widget/ConstraintLayout

[13] Espresso [23/03/2024]

https://developer.android.com/training/testing/espresso

[14] Recycler view [10/12/2023]

https://developer.android.com/develop/ui/views/layout/recyclerview

[15] SQLite [03/12/2023]

https://developer.android.com/reference/android/database/sqlite/package-summary

[16] Tested Extension [23/02/2024]

https://developer.android.com/reference/tools/gradle-api/8.0/com/android/build/api/dsl/TestedExtension

[17] Runner [13/01/2024]

https://developer.android.com/training/testing/instrumented-tests/androidx-test-libraries/runner

[18] Legacy [23/03/2024]

https://developer.android.com/jetpack/androidx/releases/legacy

[19] Activity Lifecycle [09/01/2024]

https://developer.android.com/guide/components/activities/activity-lifecycle

[20] PDF Document [10/02/2024]
https://developer.android.com/reference/kotlin/android/graphics/pdf/PdfDocument

[21] View On Click Listener [02/12/2023]

https://developer.android.com/reference/android/view/View.OnClickListener

[22] Refactor[12/12/2023]
https://subscription.packtpub.com/book/mobile/9781785286186/1/ch01lvl1sec13/refactoring-your-code

[23] Android Studio Download [03/04/2023]

https://developer.android.com/studio

[24] Layout Editor [12/10/2023]

https://developer.android.com/studio/write/layout-editor

[25] Kotlin [15/04/2023]

https://kotlinfoundation.org/

[26] Kotlin Multiplatform [20/05/2023]

https://www.jetbrains.com/kotlin-multiplatform/

[27] Kotlin Contribution [15/04/2023]

https://kotlinlang.org/docs/contribute.html

[28] Kotlin [15/04/2023]

https://devopedia.org/kotlin-language

[29] SQLite [13/11/2023]

https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase

[30] SQLite [20/11/2023]

https://www.sqlite.org/index.html

[31] W3C “Extensible Markup Language (XML)” [25/02/2024]

https://www.w3.org/XML/

[32] W3Schools “Introduction to XML” [25/02/2024]

https://www.w3schools.com/xml/xml_whatis.asp

[33] AWS “What is XML?” [25/02/2024]

https://aws.amazon.com/what-is/xml/

[34] Διπλωματική Εργασία “πολυκατοικία“, Χρονάς Άγγελος, Πανεπιστήμιο Θεσσαλίας, Τμήμα Αρχιτεκτόνων Μηχανικών Βόλος Μάρτιος 2010

https://ir.lib.uth.gr/xmlui/bitstream/handle/11615/44204/8217.pdf?sequence=

[35] JetBrains About [17/03/2024]

https://www.jetbrains.com/company/

[36] Proper [25/03/2024]

https://proper.gr

[37] Billys [25/03/2024]

https://billys.gr

[38] Draw.io [15/04/2024]

https://app.diagrams.net/

Παράρτημα κώδικα και εγκατάσταση εφαρμογής

Ο πηγαίος κώδικας της εφαρμογής βρίσκεται στο Github στον παρακάτω σύνδεσμο

https://github.com/Nousios/Maintenance_Fees_Calculator

Από το αρχείο app-debug.apk μπορεί κάποιος να κατεβάσει και να κάνει εγκατάσταση την εφαρμογή σε συσκευή Android.