



Προηγμένα Θέματα Αλληλεπίδρασης (Προγραμματισμός Κινητών Συσκευών)

Εφαρμογή για υπάλληλο e-shop

Σιαμάτρας Βασίλειος - 185273

Χαλκίδης Λάζαρος – 164773

Τα αρχεία της εφαρμογής βρίσκονται στον σύνδεσμο : [VasilisSiam/EshopAndroid \(github.com\)](https://github.com/VasilisSiam/EshopAndroid)

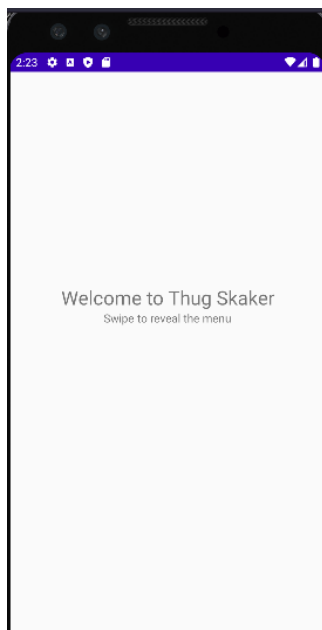
Περιεχόμενα :

- Εισαγωγή
- Databases
- Επιλογές στις τεχνολογίες που χρησιμοποιήθηκαν
- Αναλυτικά η υλοποίηση

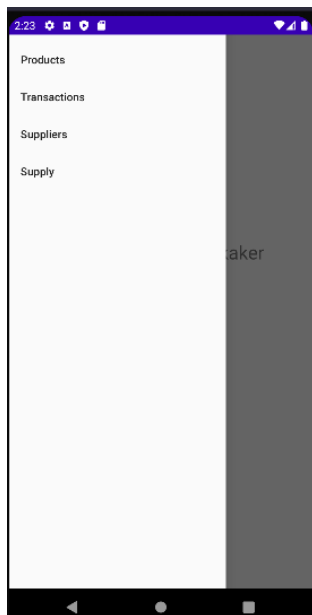
Εισαγωγή

Το θέμα της εργασίας μας ήταν η υλοποίηση ενός EShop για τον υπάλληλο του καταστήματος ,συγκεκριμένα εμάς το shop μας έχει ως λειτουργίες ο χρήστης να μπορεί να βλέπει τα προϊόντα του καταστήματος και ανάλογα να προσθέτει και να αφαιρεί ,να μπορεί να βλέπει τους προμηθευτές ,τις προμήθειες του καταστήματος και ανάλογα να προσθέτει και να αφαιρεί ,και τέλος να βλέπει αλλά και να κάνει συναλλαγές με πελάτες.

Εφαρμογή



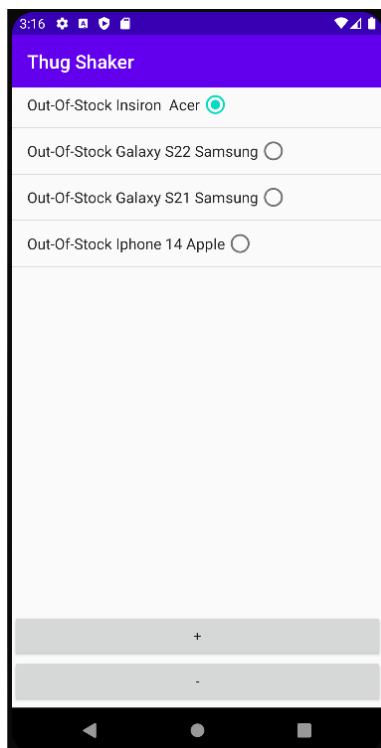
Εικόνα 1.1



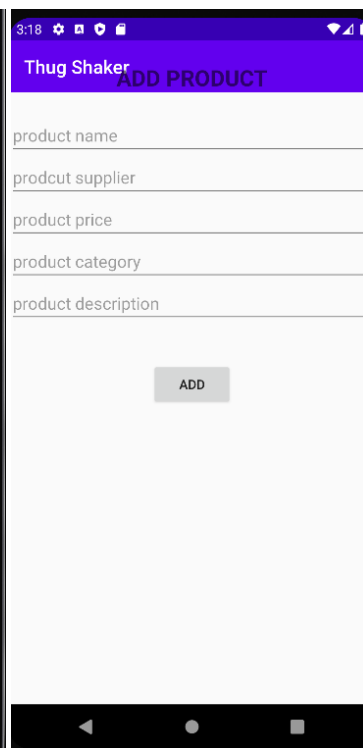
Εικόνα 1.2

Όπως παρουσιάζεται στις εικόνες 1.1 και 1.2 , η επιφάνεια εμφανίζει το Home fragment στο οποίο ο χρήστης καλείται να κάνει swiipe για να κάνει reveal το drawer menu το οποίο στοχεύει να οργανώσει καλύτερα τις λειτουργίες της εφαρμογής και από εκεί μπορεί να αλλάξει σελίδα και να μεταβεί στα επόμενα fragment.

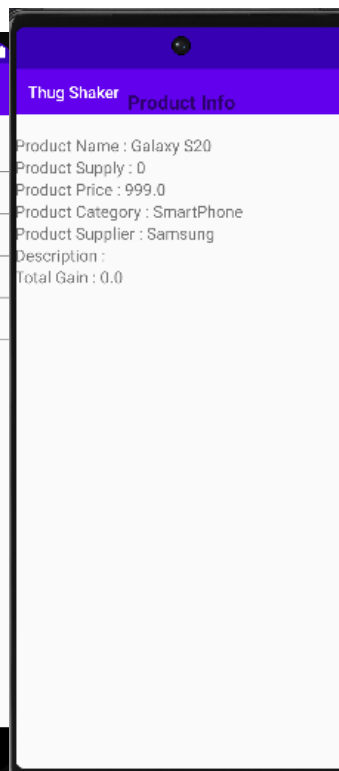
Ο χρήστης αφού κάνει κλικ στην επιλογή Products ,μεταβιβάζεται στο Fragment Products(Εικόνα 1.3) στο οποίο υπάρχουν τα κουμπιά add(+) και remove(-) όπου ο χρήστης μπορεί αντίστοιχα να προσθέσει και να αφαιρέσει προϊόντα ,παράλληλα στο ίδιο fragment υπάρχει και μια λίστα όπου εμφανίζονται τα προϊόντα του καταστήματος. Σε αυτήν την λίστα υπάρχει διπλά στο όνομα κάθε προϊόντος και μια περιγραφή που εμφανίζει αν το προϊόν είναι σε Running low,out of stock και in stock το οποίο υπολογίζετε με τρόπο ο οποίος αναλύετε παρακάτω στο section `Αναλυτικά η υλοποίηση` καθώς και ο προμηθευτής απτόν οποίο προέρχεται. Έπειτα πατώντας το κουμπί (+) ο χρήστης πηγαίνει στο fragment add Products(Εικόνα 1.4) όπου εκεί ο χρήστης μπορεί να προσθέσει ένα αντικείμενο product πατώντας το κουμπί add,αφού συμπλήρωσε τα text fields ο χρήστης και πατώντας το κουμπί add θα εμφανιστεί ένα pop up μήνυμα επιβεβαιώνοντας ότι έγινε η προσθήκη του αντικείμενου και θα του σταλεί και ένα notification .Έπειτα το καινούργιο item έχει προστεθεί στην λίστα του fragment products.Τέλος υπάρχει και του κουμπί More όπου ο χρήστης μπορεί διαλέγοντας από το list κάποιο προϊόν να το πατήσει και να μεταφερθεί στο fragment prod info όπου θα δει όλες τις πληροφορίες του προϊόντος (Εικόνα 1.5)



Εικόνα 1.3



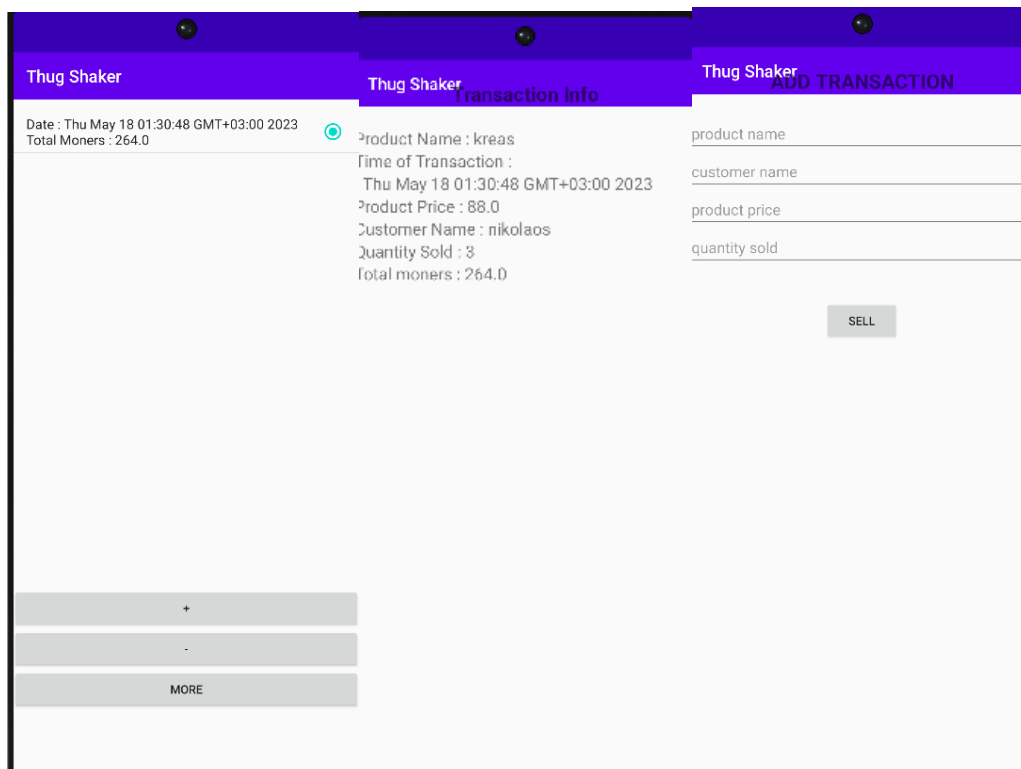
Εικόνα 1.4



Εικόνα 1.5

Στην συνέχεια το fragment Transactions(Εικόνα 1.6) όπου βλέπουμε της συναλλαγές που κάνει το κατάστημα με τον πελάτη .Στο συγκεκριμένο fragment όπως και στο products έχουμε ένα list όπου μας δείχνει της

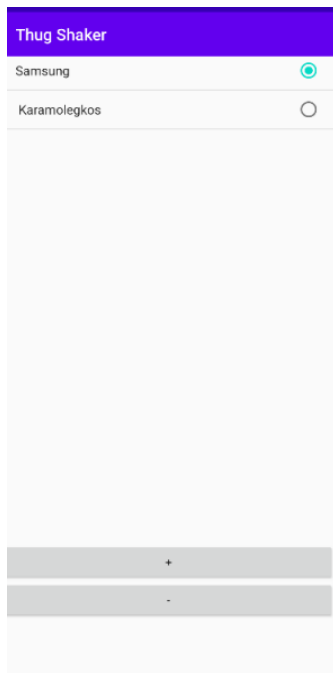
συνέλεγχος που έχουν γίνει με τις γενικές πληροφορίες που περιλαμβάνουν την στιγμή που έγινε η συναλλαγή και το συνολικό ποσό που εισπράχθηκε και έπειτα έχουμε το κουμπί (+) και το κουμπί (-) όπου αντίστοιχα μπορούμε να προσθέσουμε και να αφαιρέσουμε συναλλαγές. Πατώντας το κουμπί (+) ο χρήστης μεταφέρει στο fragment add Transactions (Εικόνα 1.8) όπου εκεί προσθέτει ο χρήστης το όνομα του Προϊόντος το όνομα του πελάτη την ποσότητα που αγόρασε και τέλος την τιμή του προϊόντος. Όπως και στο fragment products υπάρχει και εδώ το κουμπί More όπου ο χρήστης διαλέγοντας το κατάλληλο transaction μπορεί να δει όλες της πληροφορίες για το Transaction που επέλεξε (Εικόνα 1.7), σε περίπτωση που γίνει προσπάθεια να πωληθεί ένα προϊόν σε ποσότητα για την οποία δεν υπάρχει απόθεμα τότε ο υπάλληλος θα λάβει κατάλληλο notification. Υποσημείωση σχετικά με το συγκεκριμένο fragment: λόγω της χρήσης του cloud firestore το οποίο παρουσιάζει αυτό το φαινόμενο που αποκαλείτε cold start που σημαίνει ότι κάθε φορά που εκτελείτε ένα cloud function όπως read / write από την βάση για πρώτη φορά πρέπει να δημιουργηθεί ένα node με το context της εφαρμογής αλλά και την φύση των αιτημάτων που είναι ασύγχρονα, υπάρχει μια καθυστέρηση στην φόρτωση των αποτελεσμάτων και στην δυνατότητα της επεξεργασίας αυτών όπως και ο χρόνος που θα περάσει από την στιγμή που ο χρήστης θα προσπαθήσει να καταγράψει μια συναλλαγή μέχρι την στιγμή που θα δεχθεί ειδοποίηση για το αν αυτή η εγγραφή ήταν επιτυχής ειδικά την πρώτη φορά που θα μεταβεί ο χρήστης σε αυτό το fragment.



Εικόνα 1.6

Εικόνα 1.7

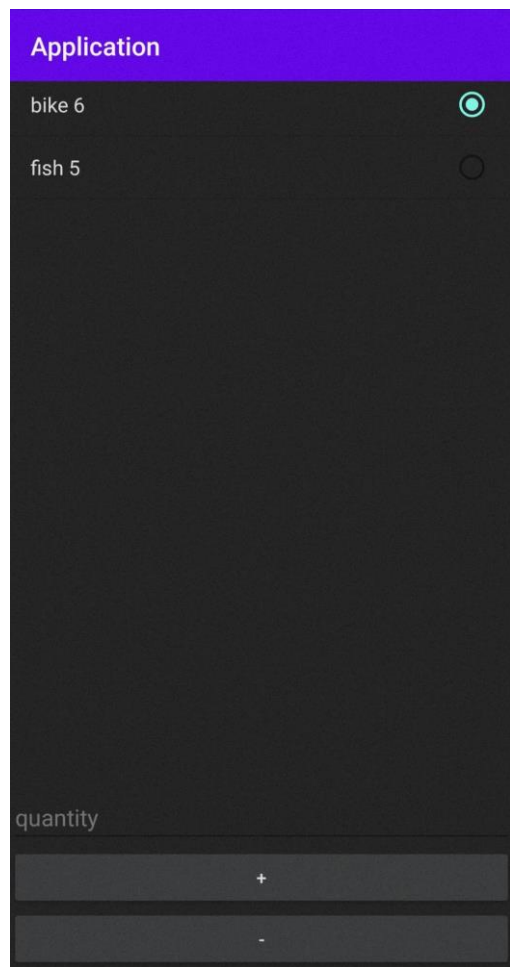
Εικόνα 1.8



Εικόνα 1.9

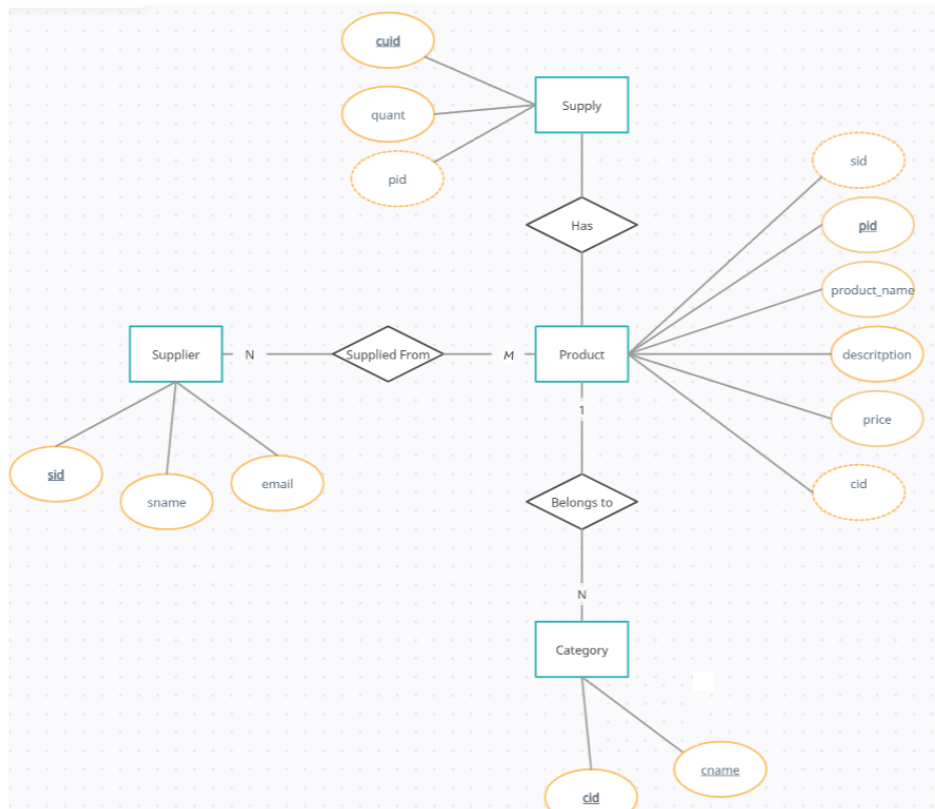
Το επόμενο Fragment είναι το Suppliers (Εικόνα 1.9) στο συγκεκριμένο fragment ο χρήστης μπορεί να προσθέσει και να αφαιρέσει Suppliers, τέλος όπως φαίνεται στην εικόνα, βλέπουμε ότι όπως και πριν τα αντικείμενα εμφανίζονται σε λίστα και το add γίνεται σε ξεχωριστό fragment επίσης αξίζει να σημειωθεί ότι σε περίπτωση που ο χρήστης κατά την εισαγωγή προϊόντος εισάγει κάποιον προμηθευτή ο οποίος δεν υπάρχει στον πίνακα τότε αυτός αυτόματος θα εισαχθεί και στον πίνακα Suppliers.

Τελευταίο είναι το Fragment Supply όπου ο χρήστης βλέπει για κάθε προϊόν ανεξάρτητα από το ποιος το προμηθευθεί τις συνολικές ποσότητες που έχει και μπορεί ανάλογα να προσθέσει και να αφαιρέσει ποσότητα η ποσότητα χρεώνετε κάθε φορά σε περίπτωση που το προϊόν διατίθεται από παραπάνω από ένα προμηθευτές σε κάποιο τυχαία και όταν αφαιρείτε αφαιρείτε συνολικά από οποίους μπορεί να αφαιρεθεί , προφανώς σε περίπτωση που ο χρήστης προσπαθήσει να αφαιρέσει παραπάνω από αυτήν που διαθέτει αυτό απαγορεύετε και του στέλνετε κατάλληλη ειδοποίηση . (Εικόνα 1.10)



Εικόνα 1.10

Databases



Εικόνα 2.1

Η τοπική βάση(διάγραμμα ER στην εικόνα 2.1) αποτελείτε από 4 πίνακες για τις κατηγορίες , προμήθεια , προμηθευτή και προϊόν με το προϊόν να παίρνει ξένα κλειδιά από την κατηγορία την οποία ανήκει και τον προμηθευτή που το προμηθεύει και την προμήθεια ξένο κλειδί από το προϊόν στο οποίο αντιστοιχεί το απόθεμα αρχικά υπήρξε το σκεπτικό ότι έχοντας data στα οποία μπορεί να χρειαστεί μαζική επεξεργασία/transformation όπως τιμή και απόθεμα στον ίδιο πίνακα θα οδηγήσει σε υψηλότερη απόδοση καθώς το struct θα είναι περισσότερο cache friendly από το να περιέχει και πληροφορίες όπως μεγάλα strings και διαφορά ξένα κλειδιά που πιθανότατα δεν χρειάζονται κατά την διάρκεια του υπολογισμού κάτι το οποίο τελικά πηρέ χαμηλότερη προτεραιότητα και δεν υλοποιήθηκε πλήρως.

customer: "nikolaos"

date: 18 May 2023 at 01:30:48 UTC+3

price: 88

product: "kreas"

quantity: 3

Εικόνα 2.2

Στην απομακρυσμένη βάση όπου αποθηκεύονται συναλλαγές, το document έχει δομή όπως επιδεικνύετε στην εικόνα 2.2 και αποθηκεύονται στο collection transactions με το πεδίο `customer` να δηλώνει το όνομα του πελάτη στον οποίον πωλείται το προϊόν, το `date` να είναι ένα timestamp που δηλώνει την ακριβή στιγμή που επικυρώθηκε η συναλλαγή, το `price` η αξία στην οποία πουλήθηκε το προϊόν, το `product` το όνομα του προϊόντος και το `quantity` η ποσότητα που αγοράστηκε από τον πελάτη, εδώ να σημειωθεί ότι οι δυο βάσεις έχουν επικοινωνία καθώς η πώληση ενός προϊόντος σε μια συγκεκριμένη ποσότητα σημαίνει ότι πρέπει να αφαιρεθεί η κατάλληλη ποσότητα αποθέματος από την τοπική βάση και αυτό επιτυγχάνετε προγραμματιστικά σε κάθε συναλλαγή.

Επιλογές στις τεχνολογίες που χρησιμοποιήθηκαν

Η εφαρμογή υλοποιήθηκε στο android studio με την χρήση του Room api και του firestore όπως προτάθηκε από τον καθηγητή και η κυρία γλώσσα ήταν η Kotlin αλλά χρησιμοποιήθηκε και Java. Κύριος λόγος που συντέλεσε στην επιλογή της Kotlin είναι ότι ο χρόνος που εξοικονομείτε λόγω του ότι δίνετε η δυνατότητα να υλοποιηθεί κάτι σε λιγότερες γραμμές κώδικα ήταν αρκετά περισσότερος από τον χρόνο που χιάστηκε για την προσαρμογή σε αυτήν την νέα γλώσσα, επίσης η overall πιο ευχάριστη εμπειρία έχοντας περισσότερες δυνατότητες στην σύνταξη της γλώσσας και τέλος ήταν μια ευκαιρία για να μάθουμε και κάτι άλλο την οποία εκμεταλλευτήκαμε καθώς η εργασία είχε έτσι κι αλλιώς εκπαιδευτικό σκοπό.

Αναλυτικά η υλοποίηση

Η εφαρμογή είχε κάποιες απαιτήσεις στην λειτουργικότητα της που έπρεπε να υλοποιηθούν με κάποιο τρόπο, αυτές ήταν: χρήση ειδοποιήσεων (Notifications) για κατάλληλη ενημέρωση του χρήστη, αυτόματη αλλαγή εμφάνισης όταν αλλάζει κατεύθυνση η συσκευή (portrait/landscape), ερωτήματα (διαφορετικής δομής – χρήση διαφορετικών τελεστών). Αναλυτικά:

```
private void createNotificationChannel() {  
    // Create the NotificationChannel, but only on API 26+ because  
    // the NotificationChannel class is new and not in the support library  
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {  
        CharSequence name = "notifications";  
        String description = "notificationDesc";  
        int importance = NotificationManager.IMPORTANCE_DEFAULT;  
        NotificationChannel channel = new NotificationChannel("notificationId", name, importance);  
        channel.setDescription(description);  
        // Register the channel with the system; you can't change the importance  
        // or other notification behaviors after this  
        NotificationManager notificationManager = getSystemService(NotificationManager.class);  
        notificationManager.createNotificationChannel(channel);  
    }  
}
```

Εικόνα 3.0

Στο κομμάτι των Notifications αρχικά δημιουργούμε με το ξεκίνημα της εφαρμογής το κανάλι όπως φαίνεται στην εικόνα 3.0 και έπειτα μέσω μιας public static μεθόδου (εικόνα 3.1) η οποία βρίσκεται στην κλάση του MainActivity(που είναι γραμμένο σε Java) ώστε να έχουν σε αυτήν πρόσβαση όλα τα fragments μας στέλνουμε το κατάλληλο μήνυμα (εφόσον ο χρήστης έχει δώσει στην εφαρμογή μας δικαίωμα να το κάνει) δίνοντας το κατάλληλο context που απαιτείται το οποίο στην Kotlin μπορεί να ανακτηθεί μέσω της μεθόδου `requireContext()`, το notification builder μπορεί να πάρει διάφορους παραμέτρους όπως το εικονίδιο που θα εμφανίζεται στην ειδοποίηση, ο ήχος που θα παίζει και αλλά πολλά εκ των οποίων δεν εμφανίζονται ούτε στον κώδικα της εφαρμογής καθώς για τις ανάγκες της εργασίας χρησιμοποιήθηκε κατά βάση το default behaviour. Το σύστημα των notifications εξυπηρετεί συγκεκριμένα στην εφαρμογή την καταλληλη ειδοποίηση του χρήστη για το ποτέ μια προσθήκη στην βάση ήταν επιτυχής, ποτέ μια πώληση ολοκληρώθηκε και ήταν επιτυχής και ποτέ προσπάθησε να κάνει πώληση μη έχοντας το προϊόν που πήγε να πουλησει.

```
public static void pushNotification(android.content.Context Self, String message) {
    if (ActivityCompat
        .checkSelfPermission(Self, android.Manifest.permission.POST_NOTIFICATIONS) != PackageManager.PERMISSION_GRANTED)
        return;

    Uri alarmSound = RingtoneManager
        .getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
    NotificationCompat.Builder builder = new NotificationCompat
        .Builder(Self, "notificationId")
        .setContentText("Application")
        .setSmallIcon(R.drawable.ic_launcher_background)
        .setAutoCancel(true)
        .setContentText(message)
        .setBadgeIconType(NotificationCompat.BADGE_ICON_LARGE)
        .setSound(alarmSound)
        .setPriority(NotificationCompat.PRIORITY_DEFAULT);

    NotificationManagerCompat managerCompat = NotificationManagerCompat.from(Self);
    managerCompat.notify(999, builder.build());
}
```

Εικόνα 3.1

```

@Entity(foreignKeys = arrayOf(ForeignKey(entity = Category::class,
parentColumns = arrayOf("cid"),
childColumns = arrayOf("cid"),
onDelete = ForeignKey.CASCADE) , ForeignKey(entity = Supplier::class,
parentColumns = arrayOf("sid"),
childColumns = arrayOf("sid"),
onDelete = ForeignKey.CASCADE)))
data class Product(
    @PrimaryKey(autoGenerate = true) val pid : Long = 0,
    @ColumnInfo(name = "cid") val cid : Int,
    @ColumnInfo(name = "sid") val sid : Int,
    @ColumnInfo(name = "product_name") val PName : String,
    @ColumnInfo(name = "price") val Price : Float,
    @ColumnInfo(name = "desc") val Desc : String?,
)

```

Εικόνα 3.2

Τον ρόλο του κύριου table της τοπικής βάσης έχει ο πίνακας Products(εικόνα 3.2) ο οποίος παίρνει ξένα κλειδιά από τον προμηθευτή και την κατηγορία στην οποία ανήκει, είναι σημαντικό να σημειωθεί ότι η συμπεριφορά του delete cascade δουλεύει πολύ καλά στο να αφαιρείται αυτόματα η κατάλληλη ποσότητα προμήθειας από τον πίνακα Supply καθώς ένα προϊόν μπορεί να υπάρχει παραπάνω από μια φορές από περισσότερους προμηθευτές και ο χρήστης όταν αλληλοεπιδρά και ψάχνει τις προμήθειες του θα πρέπει να βλέπει μόνο ανά προϊόν και ανεξάρτητα από προμηθευτή καθώς πιθανότατα δεν τον ενδιαφέρει από που προέρχεται την στιγμή που θέλει να δει εάν είναι available προς πώληση. Ο τρόπος με τον οποίο μπορεί ο χρήστης να βλέπει συνολική προμήθεια ανά προϊόν είναι ένα sum με group by product name όπως φαίνεται στην εικόνα 3.3

```

@Query("select p.product_name as product_name , sum(s.quant) as quant from supply s inner join " +
"product p on s.pid = p.pid group by product_name")
fun get_prod_supplies() : List<ProductWithQuantity>

```

Εικόνα 3.3

```

data class ProductWithQuantity(
    val product_name : String?,
    val quant : Int?
)

```

Εικόνα 3.4

Επίσης μπορεί να υπάρχει η ανάγκη για υπολογισμό συνολικού μελλοντικού κέρδους κάποιου ανά προμηθευτή οπότε απαιτητέ dot product των τιμών προϊόντων που προμηθεύει με την προμήθεια στην οποία διατίθενται όπως στην εικόνα 3.5, δηλώνοντας ένα query ως get στην Kotlin σημαίνει ότι μπορεί να κληθεί ως Property και όχι ως μέθοδος εφόσον δεν απαιτητέ κάποια παράμετρος και στην java θα μπορούσε να κληθεί κανονικά ως getter ακόμα και αν γράφτηκε ως Property σε Kotlin, βέβαια η συγκεκριμένη λειτουργία τελικά δεν βρέθηκε στο final version της εφαρμογής

```

@get:Query("select sum(p.price * s.quant) as gain , z.sname as supplier_name from supply s inner join " +
    " product p on s.pid = p.pid inner join supplier z on p.sid = z.sid " +
    " group by (p.sid) order by (gain)")
val supplier_gain : List<SupplierGain>

data class SupplierGain(
    val gain : Double?,
    val supplier_name : String?,
)

```

Εικόνα 3.5

Απαραίτητη ανάγκη είναι να αποτραπεί πιθανή πώληση προϊόντος από λάθος υπαλλήλου εφόσον δεν υπάρχει το απόθεμα για να γίνει η πώληση ή σε ακόμη πιο ακραία περίπτωση να μην υπάρχει καν αυτό το προϊόν διαθέσιμο προς πώληση, αυτό επιτυγχάνετε μέσα από ένα query(εικόνα 3.6) το οποίο εκτελείτε πριν γίνει η καταγραφή στην απομακρυσμένη βάση για να δώσει τις κατάλληλες πληροφορίες που χρειάζονται για την πώληση η να μην δώσει τίποτα σε περίπτωση που μια πώληση είναι αδύνατον να γίνει , όπως φαίνεται το query παίρνει ως παράμετρο το όνομα του προϊόντος προς πώληση και την ποσότητα η οποία πρόκειται να πουληθεί και επιστρέφει μια άδεια λίστα σε περίπτωση που το προϊόν δεν έχει άθροισμα προμήθειας από όλους τους προμηθευτές μεγαλύτερο από την ποσότητα η οποία ζητείτε , σε περίπτωση που επιστραφεί κανονικά το προϊόν σημαίνει ότι η συναλλαγή μπορεί να προχωρήσει και θα αφαιρεθεί η ποσότητα που πωλήθηκε από την συνολική προμήθεια.

```

@Query("select p.pid , p.product_name as name , p.price , s.quant from product p inner join " +
    "supply s on p.pid = s.pid where p.product_name = :name " +
    "group by p.product_name having sum(s.quant) > :ct ")
fun get_product_if_product_is_available(name : String , ct : Int ) : List<tinfo>
data class tinfo(
    val pid : Long,
    val name : String,
    val price : Float,
    val quant : Int
)

```

Εικόνα 3.6

Ένα convenience feature που στοχεύει στο να κάνει το περιβάλλον περισσότερο effective και user friendly είναι η εμφάνιση μιας κατάστασης για κάθε προϊόν που δηλώνει την διαθεσιμότητα του δηλαδή out of stock , in stock , running low , όπως φαίνεται στην εικόνα 3.8 αυτό επιτυγχάνετε με την δημιουργία ενός νέου column και την επιλογή προϊόντων με συγκεκριμένο αριθμό ποσοτήτων για κάθε μια άπαυτες τις 3 κατηγορίες χρησιμοποιώντας τον τελεστή `union` όπως φαίνεται στην εικόνα 3.7 , το query είναι πάλι μια getter.

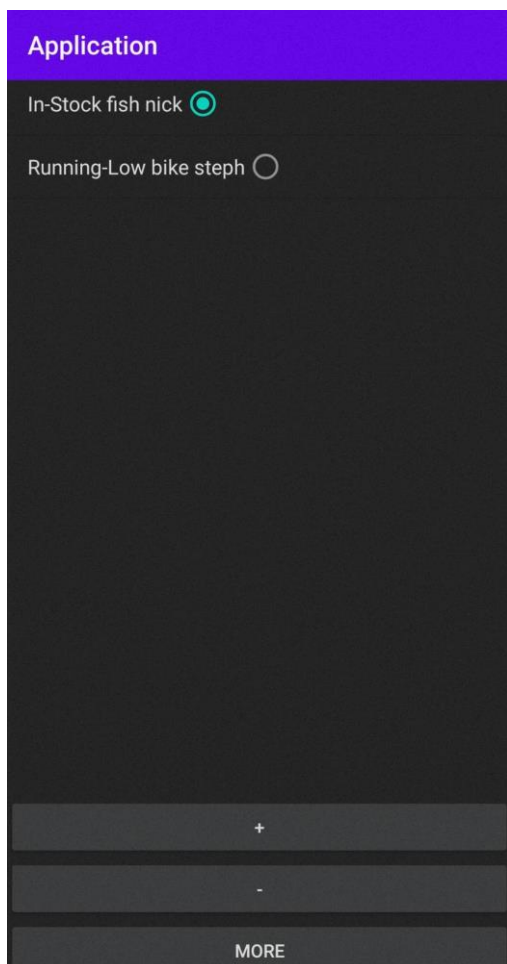
```

@get:Query("select p.pid , 'In-Stock' as availability, p.product_name as pname , s.sname " +
    "from product p join supplier s on p.sid = s.sid inner join supply z on p.pid = z.pid" +
    " where z.quant >= 4 " +
    "UNION " +
    "select p.pid , 'Out-Of-Stock' as availability, p.product_name as pname, s.sname " +
    "from product p join supplier s on p.sid = s.sid inner join supply z on p.pid = z.pid" +
    " where z.quant = 0 " +
    "UNION " +
    "select p.pid , 'Running-Low' as availability, p.product_name as pname, s.sname " +
    "from product p join supplier s on p.sid = s.sid inner join supply z on p.pid = z.pid" +
    " where z.quant < 4 and z.quant != 0 ")
val get_prods_with_supps_and_availability : List<toShow>

data class toShow(
    val pid : Long,
    val availability : String,
    val pname : String,
    val sname : String
)

```

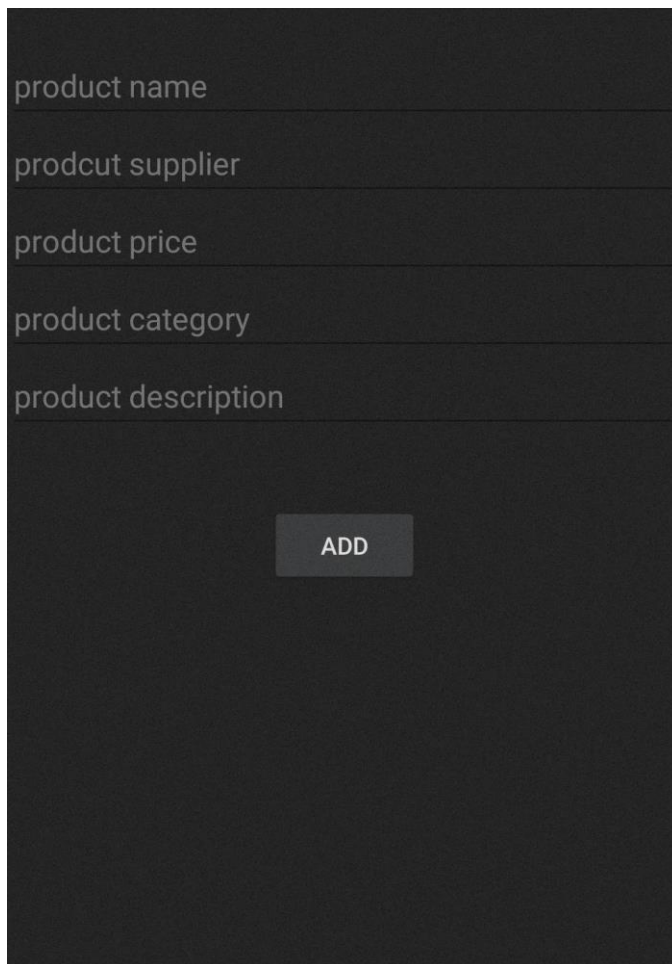
Εικόνα 3.7



Εικόνα 3.8

Η χρήση διαφορετικού layout σε portrait/landscape ήταν απαραίτητη για την σωστή λειτουργία της εφαρμογής σε ορισμένα fragments όπως το AddProduct καθώς τα περιεχόμενα δεν εμφανιζόντουσαν σωστά διαφορετικά. Αυτό επιτυγχάνετε αρκετά ευκολά βάζοντας απλά ένα landscape qualifier από το GUI του android studio και φτιάχνοντας το καινούργιο layout ώστε να

εμφανίζει το περιεχόμενο καταλληλά , η χρήση αυτού επικιρονετεμε την εμφάνιση διαφορετικού περιεχομένου στις εικόνες 3.9 , 3.10 αλλά και στις εικόνες 3.11 , 3.12 αντίστοιχα για το fragment AddTransaction που είχε παρόμοια δομή και κατ' επέκταση παρόμοια προβλήματα, η αλλαγή περιεχομένου αναφέρετε στην αύξηση του πλάτους του κουμπιού add και επίσης να σημειωθεί ότι χρειαστήκαν και άλλες αλλαγές ώστε να εμφανίζονται όλα τα περιεχόμενα χωρίς την ανάγκη ο χρήστης να πρέπει να κάνει scroll down.



product name

prodcut supplier

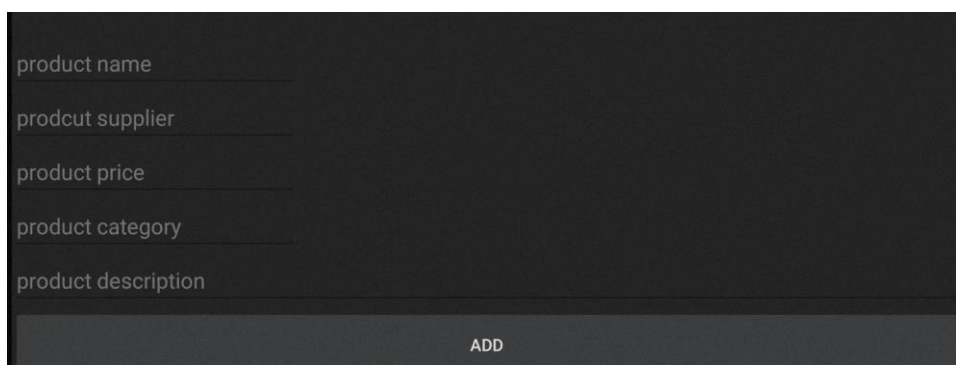
product price

product category

product description

ADD

Εικόνα 3.9



product name

prodcut supplier

product price

product category

product description

ADD

Εικόνα 3.10

product name
customer name
product price
quantity sold
<div>SELL</div>

Εικόνα 3.11

product name
customer name
product price
quantity sold
<div>SELL</div>

Εικόνα 3.12