

Contents

1. Εισαγωγή.....	2
2. Επεξήγηση Κώδικα.....	3
2.1 Main Activity	3
2.2 Adapter.....	5
2.3 Database	6
2.4 Phone Catalog.....	8
2.5 Phone Catalog DAO	9
2.5 Repository	10
2.6 View Factory	11
2.7 View Model.....	11
2.8 Xml αρχεία.....	13
3. Παρουσίαση Λειτουργιών της Εφαρμογής.....	17
3.1 Αρχική Οθόνη	17
3.2 Προσθήκη Επαφών.....	18
3.4 Διαγραφή Επαφής.....	21
3.5 Αναζήτηση Επαφής	24
4. Σύνοψη Εργασίας	27
Bibliography	28

1. Εισαγωγή

Αυτό η εργασία αφορά την κατασκευή μιας εφαρμογής διαχείρισης επαφών για κινητά τηλέφωνα για το λειτουργικό σύστημα Android, χρησιμοποιώντας τη γλώσσα Kotlin. Στόχος της εφαρμογής είναι να δώσει στους χρήστες έναν γρήγορο και απλό τρόπο να βλέπουν, να αποθηκεύουν και να αφαιρούν τις επαφές τους. Τα δεδομένα αποθηκεύτηκαν σε μια βάση δεδομένων room, εξασφαλίζοντας την ακρίβεια και την ανθεκτικότητα των εγγράφων. Επειδή η αρχιτεκτονική της εφαρμογής ακολουθεί το πρότυπο MVVM (Model-View-ViewModel), η διαχείριση των δεδομένων και η συντήρηση του κώδικα γίνονται απλά. Ένα από τα βασικά χαρακτηριστικά που δημιουργήθηκαν είναι η δυνατότητα προσθήκης νέων επαφών όπως η προβολή λίστας αποθηκευμένων επαφών με αλφαβητική ταξινόμηση μέσω μιας εύχρηστης φόρμας που ζητά το όνομα και το επώνυμο του χρήστη, τον αριθμό τηλεφώνου και τη διεύθυνση ηλεκτρονικού ταχυδρομείου(email). Επιπλέον, συμπεριλήφθηκε μια λειτουργία αναζήτησης επαφών, η οποία επιτρέπει στο χρήστη να εντοπίζει γρήγορα μια επαφή εισάγοντας τον αριθμό τηλεφώνου ή το όνομά της. Επιπλέον, η εφαρμογή επιτρέπει τη διαγραφή επαφών. Η διεπαφή χρήστη σχεδιάστηκε με προτεραιότητα στη φιλικότητα και την απλότητα, εξασφαλίζοντας μια ομαλή εμπειρία αλληλεπίδρασης. Επιπλέον, όταν τα δεδομένα στη βάση δεδομένων αλλάζουν, το LiveData χρησιμοποιείται για την αυτόματη ενημέρωση της διεπαφής χρήστη. Παρόλο που η εφαρμογή είναι λειτουργική, μπορεί να βελτιωθεί και να επεκταθεί στο μέλλον με την προσθήκη λειτουργιών η επεξεργασία επαφών κα. Κατά τη διάρκεια αυτής της διαδικασίας ανάπτυξης αποκτήθηκε σημαντική τεχνογνωσία με το MVVM, τη βάση δεδομένων Room και την ανάπτυξη μιας ολοκληρωμένης εφαρμογής Android που συνδυάζει τη χρηστικότητα, την απόδοση και την επεκτασιμότητα.

2. Επεξήγηση Κώδικα

2.1 Main Activity

Η MainActivity δημιουργεί το ViewModel χρησιμοποιώντας το ViewFactory, το οποίο παρέχει το Repository για την επικοινωνία με τη βάση δεδομένων μέσω του PhoneCatalogDao και αρχικοποιεί τη σύνδεση για τη σύνδεση των στοιχείων UI με τον κώδικα. Το RecyclerView, το οποίο συνδέεται με τον LinearLayoutManager και τον Adapter (PhoneCatalogAdapter), εμφανίζει τη λίστα των επαφών. Οι επαφές παρακολουθούνται με τη χρήση του LiveData, το οποίο ενημερώνει τη διεπαφή χρήστη κάθε φορά που αλλάζουν τα δεδομένα. Το πρόγραμμα επαληθεύει ότι τα απαραίτητα πεδία έχουν συμπληρωθεί όταν ο χρήστης επιθυμεί να προσθέσει μια νέα επαφή. Εάν ναι, χρησιμοποιείται η `viewModel.insert()` για τη δημιουργία ενός νέου αντικειμένου PhoneCatalog και την εισαγωγή του στη βάση δεδομένων. Μετά την εκκαθάριση των πεδίων κειμένου, εμφανίζεται ένα Toast με μήνυμα επιτυχίας ή σφάλματος. Ο χρήστης μπορεί να φιλτράρει τη λίστα επαφών με βάση το όνομα, το επώνυμο, τον αριθμό τηλεφώνου και το πεδίο ηλεκτρονικού ταχυδρομείου χρησιμοποιώντας το SearchView για την αναζήτηση επαφών. Το φίλτρο εφαρμόζεται χρησιμοποιώντας τη μέθοδο `filterContacts` και η φιλτραρισμένη λίστα μεταδίδεται στον Adapter για την ανανέωση της εμφάνισης της λίστας του χρήστη. Είτε οι επαφές προστίθενται, είτε αφαιρούνται, είτε αναζητούνται, ο κώδικας διασφαλίζει ότι εμφανίζονται δυναμικά ως απόκριση στις δραστηριότητες του χρήστη.

```

1 package com.example.phonecatalog
2
3 import android.os.Bundle
4 import android.widget.SearchView
5 import android.widget.Toast
6 import androidx.activity.viewModels
7 import androidx.appcompat.app.AppCompatActivity
8 import androidx.recyclerview.widget.LinearLayoutManager
9 import com.example.phonecatalog.databinding.ActivityMainBinding
10
11 class MainActivity : AppCompatActivity() {
12
13     private lateinit var binding: ActivityMainBinding
14     private val viewModel: viewModel by viewModels {
15         ViewFactory(Repository(PHONE_CATALOG_DATABASE.getDatabase(context: this).phoneCatalogDao()))
16     }
17
18     private lateinit var adapter: PhoneCatalogAdapter
19     private var contactList: List<PhoneCatalog> = emptyList()
20
21     override fun onCreate(savedInstanceState: Bundle?) {
22         super.onCreate(savedInstanceState)
23         binding = ActivityMainBinding.inflate(layoutInflater)
24         setContentView(binding.root)

```

```

24         setContentView(binding.root)
25
26         adapter = PhoneCatalogAdapter { contact -> viewModel.delete(contact) }
27         binding.recyclerView.layoutManager = LinearLayoutManager(context: this)
28         binding.recyclerView.adapter = adapter
29
30         viewModel.allContacts.observe(owner: this) { contacts ->
31             contacts?.let {
32                 contactList = it
33                 adapter.submitList(it)
34             }
35         }
36
37         // Handle Add Contact button click
38         binding.btnAddContact.setOnClickListener {
39             val firstName = binding.editFirstName.text.toString()
40             val lastName = binding.editLastName.text.toString()
41             val email = binding.editEmail.text.toString()
42             val phone = binding.editPhone.text.toString()
43
44             if (firstName.isNotBlank() && lastName.isNotBlank() && email.isNotBlank() && phone.isNotBlank()) {
45                 val newContact = PhoneCatalog(firstName = firstName, lastName = lastName, email = email, phone = phone)

```

```

45         val newContact = PhoneCatalog(firstname = firstName, lastname = lastName, email = email, phone = phone)
46         viewModel.insert(newContact)
47
48         binding.editFirstName.text.clear()
49         binding.editLastName.text.clear()
50         binding.editEmail.text.clear()
51         binding.editPhone.text.clear()
52
53         Toast.makeText(context: this, text: "Contact Added!", Toast.LENGTH_SHORT).show()
54     } else {
55         Toast.makeText(context: this, text: "Please fill all fields.", Toast.LENGTH_SHORT).show()
56     }
57 }
58
59 // Handle SearchView input
60 binding.searchView.setOnQueryTextListener(object : SearchView.OnQueryTextListener {
61     override fun onQueryTextSubmit(query: String?): Boolean {
62         filterContacts(query)
63         return true
64     }
65 }

```

```

        override fun onQueryTextChange(newText: String?): Boolean {
            filterContacts(newText)
            return true
        }
    })
}

private fun filterContacts(query: String?) {
    val filteredList = if (!query.isNullOrEmpty()) {
        contactList.filter { contact ->
            contact.firstname.contains(query, ignoreCase = true) ||
            contact.lastname.contains(query, ignoreCase = true) ||
            contact.phone.contains(query) ||
            contact.email.contains(query, ignoreCase = true)
        }
    } else {
        contactList
    }
    adapter.submitList(filteredList)
}
}

```

2.2 Adapter

Η κλάση PhoneCatalogAdapter χρησιμοποιεί τον Adapter ListAdapter για την αποτελεσματική εμφάνιση επαφών σε ένα RecyclerView, με τη βοήθεια του DiffUtil για την ανίχνευση αλλαγών στη λίστα και την ενημέρωση μόνο των απαραίτητων στοιχείων. Το onCreateViewHolder παράγει ένα PhoneCatalogViewHolder για κάθε επαφή, ενώ το onBindViewHolder συνδέει τα δεδομένα των επαφών με τα σωστά στοιχεία διάταξης. Η μέθοδος bind προσθέτει λειτουργικότητα διαγραφής μέσω του κουμπιού διαγραφής και εκχωρεί το όνομα, τον αριθμό τηλεφώνου και το email της επαφής. Το DiffUtil μπορεί να ανανεώσει αποτελεσματικά τη λίστα συγκρίνοντας επαφές και

προσδιορίζοντας αν έχουν αλλάξει χρησιμοποιώντας την κλάση ContactDiffCallback. Συνολικά, ο Adapter εγγυάται ζωντανή αλληλεπίδραση με την αφή και αποτελεσματική παρουσίαση δεδομένων.

```
package com.example.phonecatalog

import android.annotation.SuppressLint
import android.app.AlertDialog
import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.ListAdapter
import androidx.recyclerview.widget.RecyclerView
import com.example.phonecatalog.databinding.ContactItemBinding

class PhoneCatalogAdapter(private val onDeleteClick: (PhoneCatalog) -> Unit) : ListAdapter<PhoneCatalog, PhoneCatalogAdapter.PhoneCatalogViewHolder>(PhoneCatalogAdapter::PhoneCatalogViewHolder) {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): PhoneCatalogViewHolder {
        val binding = ContactItemBinding.inflate(LayoutInflater.from(parent.context), parent, attachToParent: false)
        return PhoneCatalogViewHolder(binding)
    }

    override fun onBindViewHolder(holder: PhoneCatalogViewHolder, position: Int) {
        val contact = getItem(position)
        holder.bind(contact)
    }

    inner class PhoneCatalogViewHolder(private val binding: ContactItemBinding) : RecyclerView.ViewHolder(binding.root) {

        fun bind(contact: PhoneCatalog) {
            // Bind the data to the views in the layout
            binding.contactName.text = "${contact.firstname} ${contact.lastname}"
            binding.contactPhone.text = "Phone: ${contact.phone}"
            binding.contactEmail.text = "Email: ${contact.email}"

            binding.deleteButton.setOnClickListener {
                // Δημιουργία του AlertDialog για την επιβεβαίωση
                val builder = AlertDialog.Builder(binding.root.context)
                builder.setMessage("Είστε σίγουρος ότι θέλεις να διαγράψεις αυτήν την επαφή;")
                builder.setCancelable(false)
                builder.setPositiveButton("Ναι") { dialog, id ->
                    onDeleteClick(contact) // Εκτέλεση της συνάρτησης για τη διαγραφή
                }
                builder.setNegativeButton("Όχι") { dialog, id ->
                    dialog.dismiss() // Ακύρωση της διαγραφής
                }
                val alert = builder.create()
                alert.show()
            }
        }
    }
}
```

2.3 Database

Το θεμελιώδες πλαίσιο για την αποθήκευση και τη διαχείριση των επαφών στην εφαρμογή παρέχεται από την κλάση PhoneCatalogDatabase, η οποία είναι μια υλοποίηση της βάσης δεδομένων που χρησιμοποιεί το Room. Αυτή η κλάση δηλώνεται ως βάση δεδομένων Room χρησιμοποιώντας το αναγνωριστικό Database και η οντότητα PhoneCatalog που αντιπροσωπεύει τα δεδομένα της βάσης δεδομένων παρέχεται μαζί με την έκδοση της βάσης δεδομένων (σε αυτό το παράδειγμα, έκδοση = 1).

Η μέθοδος phoneCatalogDao() επιστρέφει το PhoneCatalogDAO, το οποίο είναι το αντικείμενο πρόσβασης σε δεδομένα (DAO) που παρέχει τις λειτουργίες SQL για τη διαχείριση δεδομένων, όπως εισαγωγή, ενημέρωση, διαγραφή και ανάκτηση επαφών από τη βάση δεδομένων. Για να εξασφαλιστεί ότι υπάρχει μόνο μία ενεργή βάση δεδομένων σε ολόκληρο το πρόγραμμα, αυτή η κλάση χρησιμοποιεί το σχεδιασμό Singleton. Χρησιμοποιώντας τον Room.databaseBuilder, η

μέθοδος `getDatabase()` παράγει ή επιστρέφει την τρέχουσα βάση δεδομένων. Κάθε νήμα που προσπελαίνει τη βάση δεδομένων θα βλέπει την κατάλληλη, ενημερωμένη τιμή της `INSTANCE` αν χρησιμοποιείται η επιλογή `Volatile` για τη μεταβλητή `INSTANCE`. Όταν πολλά νήματα κάνουν αιτήσεις ταυτόχρονα, η `synchronized(this)` διασφαλίζει ότι η κατασκευή της βάσης δεδομένων ολοκληρώνεται με ασφάλεια.

```
1 package com.example.phonecatalog
2
3 import android.content.Context
4 import androidx.room.Database
5 import androidx.room.Room
6 import androidx.room.RoomDatabase
7
8 @Database(entities = [PhoneCatalog::class], version = 1, exportSchema = false)
9 abstract class PhoneCatalogDatabase : RoomDatabase() {
10
11     abstract fun phoneCatalogDao(): PhoneCatalogDAO
12
13     companion object {
14         @Volatile
15         private var INSTANCE: PhoneCatalogDatabase? = null
16
17         fun getDatabase(context: Context): PhoneCatalogDatabase {
18             return INSTANCE ?: synchronized(lock: this) {
19                 val instance = Room.databaseBuilder(
20                     context.applicationContext,
21                     PhoneCatalogDatabase::class.java,
22                     name: "phone_catalog_database"
23                 ).build()
24                 INSTANCE = instance
25             }
26         }
27     }
28 }
```

```
        INSTANCE = instance
        instance
    }
}
}
```

2.4 Phone Catalog

Προκειμένου να αποθηκεύει και να διαχειρίζεται επαφές, η κλάση PhoneCatalog χρησιμοποιεί τα χαρακτηριστικά του Room για την αναπαράσταση μιας οντότητας στη βάση δεδομένων. Το αναγνωριστικό Entity υποδεικνύει ότι αυτή η κλάση συνδέεται με τον πίνακα phoneCatalog της βάσης δεδομένων. Δεδομένου ότι το id είναι το κύριο κλειδί του πίνακα και αυτόματα αυξανόμενο (autoGenerate = true), θα αυξάνεται αυτόματα κάθε φορά που προστίθεται μια νέα επαφή, η οποία αντιστοιχεί σε μια ξεχωριστή εγγραφή. Οι στήλες του πίνακα, οι οποίες αντιπροσωπεύουν τις πληροφορίες για κάθε επαφή, είναι τα πεδία firstname, lastname, email και phone. Το αναγνωριστικό ColumnInfo, το οποίο παρέχει το όνομα της αντίστοιχης στήλης στη βάση δεδομένων, χρησιμοποιείται για τη διακόσμηση κάθε πεδίου της κλάσης. Για παράδειγμα, η βάση δεδομένων αποθηκεύει το firstname ως fname, το lastname ως lname κ.ο.κ. Αυτή η υλοποίηση εγγυάται ότι τα δεδομένα κάθε επαφής ταξινομούνται ανά πεδίο της κλάσης και αποθηκεύονται κατάλληλα στον πίνακα phoneCatalog, επιτρέποντας την αποτελεσματική και ευέλικτη αποθήκευση επαφών στη βάση δεδομένων.

```
package com.example.phonecatalog

import androidx.room.Entity
import androidx.room.PrimaryKey
import androidx.room.ColumnInfo

@Entity(tableName = "phoneCatalog")

data class PhoneCatalog(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    @ColumnInfo(name = "fname") val firstname: String,
    @ColumnInfo(name = "lname") val lastname: String,
    @ColumnInfo(name = "email") val email: String,
    @ColumnInfo(name = "phone") val phone: String
)
```


2.5 Phone Catalog DAO

Η κλάση PhoneCatalogDAO περιέχει τις βασικές μεθόδους για την αλληλεπίδραση με τον πίνακα phoneCatalog της βάσης δεδομένων. Η μέθοδος getAllContacts() ανακτά όλες τις επαφές, ταξινομημένες με βάση το πεδίο fname, επιστρέφοντας ένα LiveData για την παρακολούθηση των αλλαγών και την ενημέρωση του UI. Η getContactById(id: Int) ανακτά μια επαφή με βάση το μοναδικό αναγνωριστικό, ενώ η insert(contact: PhoneCatalog) εισάγει μια νέα επαφή, αγνοώντας τυχόν συγκρούσεις μέσω της στρατηγικής OnConflictStrategy.IGNORE. Η Delete(contact: PhoneCatalog) καταργεί τη σχετική επαφή και η update(contact: PhoneCatalog) τροποποιεί μια υπάρχουσα επαφή στον πίνακα. Κάθε μέθοδος αναστέλλεται, επιτρέποντας την ασύγχρονη εκτέλεση σε ξεχωριστό νήμα χωρίς να παρεμβαίνει στο κύριο νήμα της εφαρμογής. Οι εντολές SQL που εκτελούνται για κάθε μέθοδο καθορίζονται από σημειώσεις δωματίου (όπως Insert, Update και Delete).

```
package com.example.phonecatalog

import androidx.lifecycle.LiveData
import androidx.room.Dao
import androidx.room.Delete
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import androidx.room.Update

@Dao
interface PhoneCatalogDAO {
    @Query("SELECT * FROM phoneCatalog ORDER BY fname ASC")
    fun getAllContacts(): LiveData<List<PhoneCatalog>>

    @Query("SELECT * FROM phoneCatalog WHERE id = :id")
    suspend fun getContactById(id: Int): PhoneCatalog?

    @Insert(onConflict = OnConflictStrategy.IGNORE)
    suspend fun insert(contact: PhoneCatalog)

    @Update
    suspend fun update(contact: PhoneCatalog)

    @Delete
    suspend fun delete(contact: PhoneCatalog)
```

2.5 Repository

Η κλάση Repository λειτουργεί ως διαμεσολαβητής μεταξύ PhoneCatalogDAO και ViewModel, επιτρέποντας την απομόνωση της λογικής της βάσης δεδομένων από την υπόλοιπη εφαρμογή. Η μεταβλητή allContacts επιστρέφει ένα αντικείμενο LiveData, το οποίο παρακολουθεί τις επαφές και ενημερώνει το UI όταν αλλάζουν τα δεδομένα. Μια επαφή με το καθορισμένο id ανακτάται από τη βάση δεδομένων χρησιμοποιώντας τη συνάρτηση getContactById(id: Int). Με τη χρήση της λέξης-κλειδί suspend, οι μέθοδοι insert(contact: PhoneCatalog), update(contact: PhoneCatalog) και delete(contact: PhoneCatalog) επιτρέπουν την εισαγωγή, ενημέρωση και διαγραφή επαφών στη βάση δεδομένων. Έτσι, η κλάση προσφέρει μια ασφαλή και αποτελεσματική μέθοδο αλληλεπίδρασης με τη βάση δεδομένων χωρίς να προκαλεί την καθυστέρηση του κύριου νήματος της εφαρμογής.

```
package com.example.phonecatalog

import androidx.lifecycle.LiveData

class Repository(private val phoneCatalogDAO: PhoneCatalogDAO) {
    val allContacts: LiveData<List<PhoneCatalog>> = phoneCatalogDAO.getAllContacts()

    suspend fun getContactById(id: Int): PhoneCatalog? {
        return phoneCatalogDAO.getContactById(id)
    }

    suspend fun insert(contact: PhoneCatalog) {
        phoneCatalogDAO.insert(contact)
    }

    suspend fun update(contact: PhoneCatalog) {
        phoneCatalogDAO.update(contact)
    }

    suspend fun delete(contact: PhoneCatalog) {
        phoneCatalogDAO.delete(contact)
    }
}
```

2.6 View Factory

Μια μέθοδος για τη δημιουργία του ViewModel με τις απαραίτητες εξαρτήσεις παρέχεται από την κλάση ViewFactory, η οποία είναι μια υλοποίηση της ViewModelProvider.Factory. Σε αυτή την περίπτωση, το Repository προσφέρει τις λειτουργίες που απαιτούνται για την επικοινωνία του ViewModel με τη βάση δεδομένων. Η μέθοδος create(modelClass: Class<T>) ελέγχει αν το modelClass είναι τύπου viewModel και, αν είναι, δημιουργεί και επιστρέφει μια νέα παρουσία του viewModel, περνώντας το repository ως παράμετρο. Εάν η modelClass δεν είναι τύπου viewModel, τότε εκπέμπεται μια εξαίρεση IllegalArgumentException για να υποδείξει ότι ο τύπος ViewModel δεν υποστηρίζεται. Η χρήση του αναγνωριστικού Suppress(«UNCHECKED_CAST») γίνεται για την παράκαμψη της προειδοποίησης σχετικά με το μη ελεγχόμενο cast του viewModel(repository) στον τύπο T.

```
package com.example.phonecatalog

import androidx.lifecycle.ViewModel
import androidx.lifecycle.ViewModelProvider

class ViewFactory(private val repository: Repository) : ViewModelProvider.Factory {
    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(viewModel::class.java)) {
            @Suppress("...names: \"UNCHECKED_CAST\"")
            return viewModel(repository) as T
        }
        throw IllegalArgumentException("Unknown ViewModel class")
    }
}
```

2.7 View Model

Σύμφωνα με τον κύκλο ζωής της εφαρμογής, η κλάση viewModel διαχειρίζεται τα δεδομένα της εφαρμογής επεκτείνοντας το viewmodel. Προκειμένου να λαμβάνει επαφές από τη βάση δεδομένων και να εκτελεί δραστηριότητες CRUD (εισαγωγή, ενημέρωση, διαγραφή), η κλάση αυτή αλληλεπιδρά με το repository. Το UI μπορεί να παρακολουθεί τις αλλαγές στις επαφές και να ενημερώνεται άμεσα όταν αλλάζουν τα δεδομένα χάρη στη μεταβλητή allContacts, η οποία περιέχει τα LiveData που παρέχονται από το Repository. Για την ανάκτηση μιας επαφής με βάση το id, η συνάρτηση αναστολής getContactById(id: Int) καλεί τη σχετική μέθοδο του αποθετηρίου. Οι αντίστοιχες ενέργειες CRUD εκτελούνται ασύγχρονα χωρίς να προκαλούν μπλοκάρισμα του κύριου νήματος της εφαρμογής από τις κανονικές συναρτήσεις insert(contact: PhoneCatalog), update(contact: PhoneCatalog) και delete(contact: PhoneCatalog) μέσω της viewModelScope.launch. Προκειμένου να διατηρηθεί η αποδοτικότητα του κώδικα και η ασφάλεια, το viewModelScope είναι υπεύθυνο για τον τερματισμό των λειτουργιών όταν καταστρέφεται το ViewModel.

```
package com.example.phonecatalog

import androidx.lifecycle.LiveData
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import kotlinx.coroutines.launch

class viewModel(private val repository: Repository) : ViewModel() {

    val allContacts: LiveData<List<PhoneCatalog>> = repository.allContacts

    suspend fun getContactById(id: Int): PhoneCatalog? {
        return repository.getContactById(id)
    }

    fun insert(contact: PhoneCatalog) {
        viewModelScope.launch {
            repository.insert(contact)
        }
    }

    fun update(contact: PhoneCatalog) {
        viewModelScope.launch {
            repository.update(contact)
        }
    }

    fun delete(contact: PhoneCatalog) {
        viewModelScope.launch {
            repository.delete(contact)
        }
    }
}
```

2.8 Xml αρχεία

Η κύρια διεπαφή χρήστη της εφαρμογής σχεδιάστηκε χρησιμοποιώντας την υλοποίηση `activity_main.xml`, η οποία φαίνεται στην παρακάτω εικόνα.

Όλα τα στοιχεία που απαιτούνται για την απρόσκοπτη πλοήγηση και αλληλεπίδραση των χρηστών περιλαμβάνονται σε αυτήν τη διάταξη, συμπεριλαμβανομένου ενός κουμπιού για την προσθήκη νέων επαφών, ενός `RecyclerView` για την εμφάνιση των επαφών και κατάλληλες ρυθμίσεις περιθωρίου και διαστήματος που εγγυώνται μια θετική εμπειρία χρήστη.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <!-- Section for Adding a New Contact -->
    <EditText
        android:id="@+id/editFirstName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="First Name"
        android:inputType="text" />

    <EditText
        android:id="@+id/editLastName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Last Name"
        android:inputType="text" />

    <EditText
        android:id="@+id/editEmail"
```

```

<EditText
    android:id="@+id/editEmail"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Email"
    android:inputType="textEmailAddress" />

<EditText
    android:id="@+id/editPhone"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Phone"
    android:inputType="phone" />

<Button
    android:id="@+id/btnAddContact"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Add Contact"
    android:layout_marginTop="8dp"/>

<!-- SearchView for filtering contacts -->
<SearchView
    android:id="@+id/searchView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:queryHint="Search Contact"
    android:layout_marginTop="16dp"/>

<!-- RecyclerView to display the list of contacts -->
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recyclerView"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_marginTop="16dp"
    android:layout_weight="1" />
</LinearLayout>

```

Παρακάτω απεικονίζεται μια εικόνα με την υλοποίηση της **contact_item.xml**, όπου σχεδιάστηκε η διάταξη για την εμφάνιση κάθε επαφής στη λίστα.

```
        android:text="Phone: 1234567890" />

        <TextView
            android:id="@+id/contactEmail"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Email: john.doe@example.com" />

        <Button
            android:id="@+id/deleteButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="8dp"
            android:text="Delete" />

    </LinearLayout>
</androidx.cardview.widget.CardView>

        android:text="Phone: 1234567890" />

        <TextView
            android:id="@+id/contactEmail"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Email: john.doe@example.com" />

        <Button
            android:id="@+id/deleteButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="8dp"
            android:text="Delete" />

    </LinearLayout>
</androidx.cardview.widget.CardView>
```

Παρακάτω απεικονίζεται μια εικόνα με την υλοποίηση της **item_contact.xml**, όπου σχεδιάστηκε η διάταξη για την εμφάνιση μεμονωμένων επαφών στη λίστα. Περιλαμβάνει πεδία όπως όνομα, επώνυμο, email και αριθμό τηλεφώνου, διαμορφωμένα ώστε να παρέχουν μια καθαρή και οργανωμένη παρουσίαση.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp"
    android:padding="16dp"
    android:radius="8dp">

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="start">

        <!-- Full Name (First Name and Last Name) -->
        <TextView
            android:id="@+id/textViewFullName"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Full Name"
            android:textSize="18sp"
            android:textStyle="bold"
            android:paddingBottom="4dp"
            android:textColor="@android:color/black" />

            android:textColor="@android:color/black" />

        <!-- Email -->
        <TextView
            android:id="@+id/textViewEmail"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Email"
            android:textSize="14sp"
            android:textColor="@android:color/darker_gray"
            android:paddingBottom="4dp" />

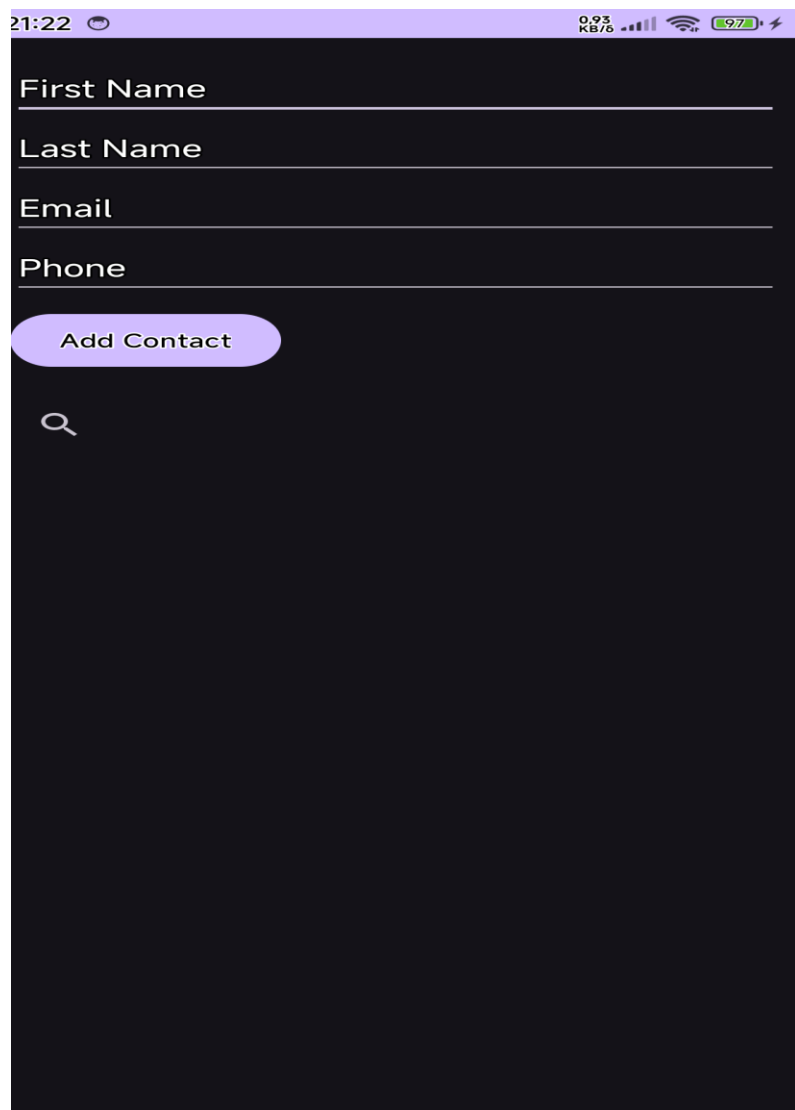
        <!-- Phone Number -->
        <TextView
            android:id="@+id/textViewPhone"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Phone Number"
            android:textSize="14sp"
            android:textColor="@android:color/darker_gray" />

    </LinearLayout>
</androidx.cardview.widget.CardView>
```


3. Παρουσίαση Λειτουργιών της Εφαρμογής

3.1 Αρχική Οθόνη

Η εφαρμογή εμφανίζει μια φόρμα για την προσθήκη επαφών. Υπάρχουν τέσσερα πεδία εισαγωγής για το "First Name" (Όνομα), "Last Name" (Επώνυμο), "Email" και "Phone" (Τηλέφωνο). Κάτω από τα πεδία υπάρχει ένα κουμπί με την ετικέτα "Add Contact" (Προσθήκη Επαφής). Επιπλέον, υπάρχει ένα εικονίδιο αναζήτησης κάτω αριστερά. Το φόντο της εφαρμογής είναι επίτηδες σκούρο, και τα στοιχεία έχουν ανοιχτές αποχρώσεις για καλή αντίθεση.



The screenshot displays the main interface of the application on a mobile device. At the top, a status bar shows the time as 21:22, signal strength, and battery level at 97%. The app's interface has a dark background. It features four text input fields stacked vertically, labeled "First Name", "Last Name", "Email", and "Phone". Below these fields is a prominent red rounded rectangular button labeled "Add Contact". At the bottom left of the screen, there is a magnifying glass icon for search functionality.

3.2 Προσθήκη Επαφών

Η δυνατότητα προσθήκης επαφών σε μια εφαρμογή διαχείρισης επαφών είναι θεμελιώδης για την ενημέρωση και οργάνωση του καταλόγου. Όπως παρατηρούμε, η εφαρμογή απαιτεί τη συμπλήρωση τεσσάρων βασικών στοιχείων για κάθε επαφή: όνομα, επώνυμο, ηλεκτρονικό ταχυδρομείο και τηλέφωνο. Όταν βεβαιωθούμε ότι τα στοιχεία που έχουμε εισάγει είναι σωστά, πατάμε το κουμπί «Add Contact». Αν η εγγραφή γίνει σωστά, θα εμφανιστεί το μήνυμα «Contact Added!» και η νέα επαφή θα εμφανιστεί στο κέντρο της οθόνης. Αν όμως παραλείψουμε κάποιο από τα απαιτούμενα στοιχεία, το μήνυμα «Please fill all fields» θα μας υπενθυμίσει να το συμπληρώσουμε. Οι εγγραφές καταχωρούνται πάντα σε αριθμητική σειρά, σύμφωνα με το όνομα των επαφών, πράγμα που βοηθά στη σωστή οργάνωση και στην εύκολη εύρεση των επαφών αργότερα. Η διαδικασία αυτή εξασφαλίζει ότι ο χρήστης μπορεί να προσθέτει νέες επαφές γρήγορα και με ασφάλεια, χωρίς να χρειάζεται να ανησυχεί για τυχόν λάθη ή παραλείψεις στην καταχώρηση των στοιχείων.

First Name

ΠΑΥΛΟΣ

Last Name

ΚΟΛΟΒΟΣ

Email

kolovospaul@gmail.com

Phone

6945711895

Add Contact

Add Contact



ΠΑΥΛΟΣ ΚΟΛΟΒΟΣ

Phone: 6945711895

Email: kolovospaul@gmail.com

Delete



Contact Added!

First Name

Last Name

Email

Phone

Add Contact



ΑΝΔΡΕΑΣ ΠΑΠΑΔΟΠΟΥΛΟΣ

Phone: 6966072000

Email: andreaspapadopoulos@gmail.com

Delete

ΠΑΥΛΟΣ ΚΟΛΟΒΟΣ

Phone: 6945711895

Email: kolovospaul@gmail.com

Delete

ΡΟΥΜΕΛΙΩΤΗΣ ΓΙΩΡΓΟΣ

Phone: 6945780999

Email: roumeliotis@gmail.com

Delete

3.4 Διαγραφή Επαφής

Η δυνατότητα αφαίρεσης επαφών από το πρόγραμμά μας είναι εξίσου σημαντική για την οργάνωση της λίστας. Το πρόγραμμα μας δίνει τη δυνατότητα να διαγράψουμε γρήγορα επαφές όταν τις έχουμε προσθέσει και θέλουμε να απαλλαγούμε από κάποιες από αυτές. Έχουμε ήδη προσθέσει τρεις επαφές, όπως φαίνεται στην εικόνα, και είναι καταχωρημένες με βάση το όνομα τους με αλφαβητική σειρά. Σε περίπτωση που επιλέξουμε να αφαιρέσουμε μία από αυτές, κάνουμε κλικ στο κουμπί «Διαγραφή» στη σχετική επαφή. Η εφαρμογή μας προτρέπει αμέσως με την ερώτηση «Είστε σίγουροι ότι θέλετε να διαγράψετε αυτή την επαφή;» για να βεβαιωθούμε ότι είμαστε σίγουροι με τη διαγραφή. Η επιλογή «Ναι» έχει ως αποτέλεσμα τη μόνιμη διαγραφή της επαφής από τη λίστα. Οι εικόνες δείχνουν το πριν και το μετά της διαδικασίας: στην αρχή εμφανίζονται τρεις επαφές, αλλά μετά τη διαγραφή απομένουν μόνο δύο. Χωρίς να επηρεαστεί ο κατάλογος, η επιλογή «Όχι» ακυρώνει τη διαγραφή και επαναφέρει την αρχική κατάσταση με τις τρεις επαφές. Η διαδικασία αυτή δίνει στους χρήστες πλήρη έλεγχο των επαφών τους και τους βοηθά να διατηρούν έναν τακτοποιημένο κατάλογο.

<div>First Name</div> <div>Last Name</div> <div>Email</div> <div>Phone</div> <div>Add Contact</div> <div><div></div></div> <div><div>ΑΝΔΡΕΑΣ ΠΑΠΑΔΟΠΟΥΛΟΣ</div><div>Phone: 6966072000</div><div>Email: andreaspapadopoulos@gmail.com</div><div>Delete</div></div> <div><div>ΠΑΥΛΟΣ ΚΟΛΟΒΟΣ</div><div>Phone: 6945711895</div><div>Email: kolovospaul@gmail.com</div><div>Delete</div></div> <div><div>ΡΟΥΜΕΛΙΩΤΗΣ ΓΙΩΡΓΟΣ</div><div>Phone: 6945780999</div><div>Email: roumeliotis@gmail.com</div><div>Delete</div></div>	<div>First Name</div> <div>Last Name</div> <div>Email</div> <div>Phone</div> <div>Add Contact</div> <div><div></div></div> <div><div>ΑΝΔΡΕΑΣ ΠΑΠΑΔΟΠΟΥΛΟΣ</div><div>Eίσαι σίγουρος ότι θέλεις να διαγράψεις αυτήν την επαφή;</div><div><div>OXI</div><div>NAI</div></div></div> <div><div>ΠΑΥΛΟΣ ΚΟΛΟΒΟΣ</div><div>Phone: 6945711895</div><div>Email: kolovospaul@gmail.com</div><div>Delete</div></div> <div><div>ΡΟΥΜΕΛΙΩΤΗΣ ΓΙΩΡΓΟΣ</div><div>Phone: 6945780999</div><div>Email: roumeliotis@gmail.com</div><div>Delete</div></div>
--	--

First Name

Last Name

Email

Phone

Add Contact



ΑΝΔΡΕΑΣ ΠΑΠΑΔΟΠΟΥΛΟΣ

Phone: 6966072000

Email: andreaspapadopoulos@gmail.com

Delete

ΠΑΥΛΟΣ ΚΟΛΟΒΟΣ

Phone: 6945711895

Email: kolovospaul@gmail.com

Delete

3.5 Αναζήτηση Επαφής

Για τον γρήγορο εντοπισμό ορισμένων επαφών στον κατάλογο, η λειτουργία αναζήτησης επαφών της εφαρμογής μας είναι πραγματικά χρήσιμη. Όταν πατήσετε το εικονίδιο του μεγεθυντικού φακού της εφαρμογής, ενεργοποιείται η λειτουργία αναζήτησης. Όταν κάνουμε κλικ στο εικονίδιο, εμφανίζεται το κλασικό πεδίο αναζήτησης «Search Contact». Για παράδειγμα, η εφαρμογή αρχίζει να φιλτράρει τις επαφές και εμφανίζει μόνο εκείνες που ταιριάζουν με το όνομα «ΠΑΥΛΟΣ» αν το πληκτρολογήσουμε με κεφαλαία γράμματα στην περιοχή αναζήτησης. Δεδομένου ότι υπάρχουν μόνο δύο επαφές σε αυτή την περίπτωση και η μία από αυτές ονομάζεται «ΠΑΥΛΟΣ», η πληκτρολόγηση «ΠΑΥΛΟΣ» θα αποκαλύψει μόνο την επαφή που ταιριάζει με αυτό το όνομα, όπως φαίνεται στην εικόνα. Υπάρχει επίσης ένα μικρό «X» στη δεξιά πλευρά του πλαισίου αναζήτησης. Πρέπει μόνο να κάνουμε κλικ στο «X» για να καταργήσουμε την αναζήτηση και να επιστρέψουμε σε ολόκληρη τη λίστα. Με αυτόν τον τρόπο, θα διαγράψουμε τις λέξεις που πληκτρολογήσαμε και το αποτέλεσμα της αναζήτησης θα εξαφανιστεί, επαναφέροντας τον κατάλογο στην αρχική του κατάσταση. η εφαρμογή είναι πιο εύχρηστη και φιλική προς το χρήστη λόγω της λειτουργίας αναζήτησης, η οποία επιτρέπει στους χρήστες να εντοπίζουν γρήγορα ορισμένες επαφές.

First Name

Last Name

Email

Phone

Add Contact



ΑΝΔΡΕΑΣ ΠΑΠΑΔΟΠΟΥΛΟΣ

Phone: 6966072000

Email: andreaspapadopoulos@gmail.com

Delete

ΠΑΥΛΟΣ ΚΟΛΟΒΟΣ

Phone: 6945711895

Email: kolovospaul@gmail.com

Delete

First Name

Last Name

Email

Phone

Add Contact

| 🔍 Search Contact



ΑΝΔΡΕΑΣ ΠΑΠΑΔΟΠΟΥΛΟΣ

Phone: 6966072000

Email: andreaspapadopoulos@gmail.com

Delete

ΠΑΥΛΟΣ ΚΟΛΟΒΟΣ

Phone: 6945711895

Email: kolovospaul@gmail.com

Delete

First Name

Last Name

Email

Phone

Add Contact

ΠΑΥΛΟΣ



ΠΑΥΛΟΣ ΚΟΛΟΒΟΣ

Phone: 6945711895

Email: kolovospaul@gmail.com

Delete

4. Σύνοψη Εργασίας

Η εργασία περιγράφει τον τρόπο χρήσης της αρχιτεκτονικής MVVM και της γλώσσας Kotlin για τη δημιουργία μιας εφαρμογής διαχείρισης επαφών Android. Οι χρήστες μπορούν να προσθέτουν, να προβάλλουν, να αφαιρούν και να αναζητούν επαφές χρησιμοποιώντας την εφαρμογή. Η βάση δεδομένων Room αποθηκεύει τα δεδομένα με αποτελεσματικό και ασφαλές τρόπο, ενώ η διεπαφή χρήστη δημιουργήθηκε για να είναι εύχρηστη και διασκεδαστική. Βασικές λειτουργίες όπως η αλφαβητική ταξινόμηση και η ζωντανή ανανέωση της οθόνης μέσω του LiveData υλοποιούνται από την εφαρμογή. Το πρόγραμμα μπορεί να έχει βελτιωθεί με την προσθήκη πρόσθετων λειτουργιών ή τη βελτίωση της διαχείρισης επαφών. Η δημιουργία του ήταν μια αξιοσημείωτη εμπειρία, ιδίως όσον αφορά την ενσωμάτωση του Room για την αποθήκευση δεδομένων και τη χρήση του MVVM, και βελτίωσε επίσης την κατανόηση της ανάπτυξης εφαρμογών Android.

Bibliography

ier, R. (2013). **Professional Android 4 Application Development** (3rd ed.).

Wrox.Δ.Γαβαλάς, Β. Κασαπάκης, Θ. Χατζηδημήτρης, *Κινητές Τεχνολογίες*, 1η έκδοση, 2015,Εκδόσεις Νέων Τεχνολογιών ΙΚΕ.

Wangereka, H. (2023).*Mastering Kotlin for Android 14: Build powerful Android apps fromscratch using Jetpack libraries and Jetpack Compose*. Packt Publishing

Saumont, P.-Y. (2020).*The Joy of Kotlin*. Leanpub