Mobilprogramozás jegyzőkönyv

MyShoppingList

Nyakó Sándor

PHRO7R

Miskolc, 2026. január 21.

Tartalomjegyzék

1. Bevezetés

A **MyShoppingList** egy Android-alapú alkalmazás, amely a felhasználók számára hatékony eszközt biztosít a bevásárlólisták rendszerezésére és nyomon követésére. Az alkalmazás támogatja a listaelemek létrehozását, módosítását és törlését, lehetőséget ad kategóriák szerinti szűrésre és keresésre, valamint statisztikák megjelenítésére a becsült és tényleges kiadások alakulásáról.

A helyi adatok tárolását Room adatbázis biztosítja. Az alkalmazás felépítése az MVVM architektúrát követi, megjelenésében a Material Design irányelveire épül, míg az adatkezelés és az állapotváltozások reaktív módon, Kotlin Coroutines és Flow használatával valósulnak meg.

2. Projekt áttekintés

2.1 Funkcionális követelmények

2.1.1 Bevásárlólistaelemek kezelése

- Elem hozzáadása: Cím, mennyiség, ár, kategória és jegyzetek megadásával
- Szerkesztés: Meglévő bejegyzések módosítása
- Törlés: Bejegyzések eltávolítása swipe-to-delete funkcióval vagy menüből
- Vásárolt státusz: Checkbox-szal jelölhető vásárolt státusz, amely vizuálisan is megjelenik (áthúzás, átlátszóság)
- Kategóriák: Étel, Ital, Háztartás, Személyes, Egyéb kategóriák közül választható

2.1.2 Keresés és szűrés

- Keresés: Cím vagy jegyzetek alapján keresés
- Szűrés: Kategória szerinti szűrés chip gombokkal
- Rendezés: Dátum, értékelés, cím alapján rendezés
- Valós idejű frissítés: A lista automatikusan frissül a szűrők és keresés változásakor

2.1.3 Statisztikák és megosztás

- Statisztikák megjelenítése: Összes elem száma, függőben lévő és vásárolt elemek száma
- Költség kalkuláció: Elköltött összeg (vásárolt elemek) és becsült összeg (összes elem)
- Lista megosztása: Formázott szöveg formában megosztás más alkalmazásokkal

2.2 Nem funkcionális követelmények

A rendszer nem-funkcionális követelményei négy fő területre oszlanak: teljesítmény, használhatóság, biztonság és karbantarthatóság. A teljesítmény szempontjából az alkalmazásnak gyorsan, két másodpercen belül be kell töltődnie, amelyet optimalizált adatbázis lekérdezések és reaktív adatfolyamok biztosítanak. A használhatóságot egy intuitív, reszponzív felhasználói felület garantálja, amely mobilon és tableten is jól működik, egyértelmű navigációval és vizuális visszajelzésekkel (Snackbar, animációk).

A biztonsági követelmények között szerepel az adatok helyi tárolása Room adatbázisban, a beépített SQL injection védelem a Room által biztosított paraméterezett lekérdezések révén, valamint az inputok megfelelő validálása. A karbantarthatóság érdekében az alkalmazás a Clean Code és SOLID elveket követi, Kotlin típusrendszert használ, valamint moduláris MVVM architektúrával épül fel, hogy a jövőbeli bővítések egyszerűen megvalósíthatók legyenek.

3. Technológiai Stack

3.1 Core technológiák

3.1.1 Kotlin

A Kotlin modern, típusbiztos programozási nyelv, amely teljes mértékben kompatibilis a Java-val, miközben számos előnyt nyújt: null-safety, extension functions, data classes, coroutines támogatás. A nyelv rövidebb és olvashatóbb kódot tesz lehetővé, miközben a teljesítmény megegyezik a Java teljesítményével. A kiváló IDE támogatás (Android Studio) és a Google hivatalos támogatása biztosítja a hosszú távú fenntarthatóságot.

Használt Kotlin funkciók:

- Data classes – adatmodell definíciókhoz (ShoppingItem)
- Sealed classes – típusbiztos állapotkezeléshez
- Extension functions – kényelmi függvények
- Coroutines – aszinkron műveletek kezelése
- Flow – reaktív adatfolyamok
- Lambda expressions – funkcionális programozás

3.1.2 Android SDK

Az Android SDK-t használjuk az alkalmazás alapvető funkcionalitásához. A minimális SDK verzió 24 (Android 7.0), amely lehetővé teszi a modern Android funkciók használatát, miközben még mindig támogatja a régebbi eszközök jelentős részét. A target SDK 36 (Android 14), amely biztosítja a legújabb Android funkciók és biztonsági javítások használatát

3.1.3 MVVM architektúra

A Model-View-ViewModel architektúrát azért választottam, mert lehetővé teszi a kód szétválasztását és tesztelhetőségét. A View (Activity, Layout) csak a UI-t kezeli, a ViewModel az üzleti logikát, míg a Model (Repository, Database) az adatkezelést. Ez a struktúra megkönnyíti a karbantartást és a bővítést.

3.2 Adatkezelés

3.2.1 Room Database

A room egy SQLite absztrakciós réteg, amely lehetővé teszi a típusbiztos adatbázis hozzáférést minimális boilerplate kóddal. A Room automatikusan generálja a szükséges kódot a DAO interfészekből és entitásokból, valamint biztosítja a compile-time ellenőrzést.

Használt Room funkciók:

- @Entity – adatbázis táblák definíciója
- @Dao – adatbázis műveletek interfészei
- @Database – adatbázis konfiguráció
- Flow – reaktív lekérdezések automatikus frissítéssel
- TypeConverters – egyedi típusok konverziója (Enum)

3.2.2 Kotlin Coroutines és Flow

A Coroutines lehetővé teszik az aszinkron műveletek írását szinkron kód stílusban, miközben a Flow reaktív adatfolyamokat biztosít, amelyek automatikusan értesítik a figyelőket változásokkor. Ez kombináció ideális az adatbázis műveletekhez és a UI frissítésekhez.

Használt Coroutines/Flow funkciók:

- suspend functions – aszinkron adatbázis műveletek
- StateFlow – állapotkezelés reaktív módon
- combine() – több Flow kombinálása
- stateIn() – Flow StateFlow-vá alakítása
- lifecycleScope – lifecycle-aware coroutine scope

3.3 UI és Styling

3.3.1 Material Design Components

Material design Components könyvtárat használjuk a modern, konzisztens UI elemekhez. Ez biztosítja a Google Material Design irányelveinek követését és egy professzionális megjelenést.

Használt Material komponensek:

- MaterialCardView – kártyák a statisztikáknak és lista elemeknek
- MaterialToolbar – app bar a címmel
- TextInputLayout – modern input mezők outline stílussal
- ChipGroup – kategória kiválasztás chipekkel
- FloatingActionButton – új elem hozzáadása gomb
- Snackbar – visszajelzések a felhasználónak

3.3.2 View Binding

A View Binding típusbiztos hozzáférést biztosít a layout fájlok view elemeihez, megszüntetve a findViewById() hívásokat és a runtime NullPointerException kockázatát. A binding osztályok automatikusan generálódnak a layout fájlokból.

4. Projekt Struktúra

4.1 MainActivity.kt – belépési pont és UI kezelés

A MainActivity az alkalmazás fő Activity osztálya, amely kezeli a felhasználói felületet és a felhasználói interakciókat. Az Activity inicializálja az összes szükséges komponenst: nézeteket, ViewModel-t, RecyclerView-t, swipe funkciót, eseménykezelőket és adatfigyelőket.

Főbb metódusok:

onCreate() – Az Activity inicializálása:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    initializeViews()
    initializeViewModel()
    setupRecyclerView()
    setupSwipeToDelete()
    setupEventListeners()
    observeData()
}
```

Az onCreate metódus inicializálja az összes szükséges komponenst: nézeteket, ViewModel-t, RecyclerView-t, swipe funkciót, eseménykezelőket és adatfigyelőket.

setupRecyclerView() – Lista inicializálása:

```
1 Usage
private fun setupRecyclerView() {
    adapter = ShoppingItemAdapter(
        onToggleBought = { item, checked -> viewModel.toggleBought(item, isBought = checked) },
        onItemClick = { item -> showEditDeleteDialog(item) }
    )

    binding.rvItems.apply {
        layoutManager = LinearLayoutManager( context = this@MainActivity)
        adapter = this@MainActivity.adapter
        setHasFixedSize(true)
    }
}
```

Létrehozza a RecyclerView adaptert callback függvényekkel, amelyek kezelik a checkbox állapotváltását és az elemre kattintást. Beállítja a lineáris elrendezést.

setupSwipeToDelete() – Swipe törlés funkció:

```
1 Usage
private fun setupSwipeToDelete() {
    val swipeCallback = object : SwipeToDeleteCallback( context = this) {
        override fun onSwiped(viewHolder: androidx.recyclerview.widget.RecyclerView.ViewHolder, direction: Int) {
            val position = viewHolder.adapterPosition
            if (position == androidx.recyclerview.widget.RecyclerView.NO_POSITION) return

            val item = adapter.getItemAt(position)
            viewModel.deleteItem(item)

            Snackbar.make( view = binding.root, text = "${item.name} deleted", duration = Snackbar.LENGTH_LONG)
                .setAction( text = "UNDO") { viewModel.insertItem(item) }
                .setAnchorView(binding.btnAdd)
                .show()
        }
    }
    ItemTouchHelper(swipeCallback).attachToRecyclerView( recyclerView = binding.rvItems)
}
```

Implementálja a swipe-to-delete funkciót. Amikor a felhasználó balra húzza egy elemet, az törlődik, és egy Snackbar jelenik meg visszavonási lehetőséggel.

observeItems() – Adatok figyelése:

```
1 Usage
private fun observeItems() {
    lifecycleScope.launch {
        repeatOnLifecycle( state = androidx.lifecycle.Lifecycle.State.STARTED) {
            viewModel.items.collect { items ->
                adapter.submitList( newItems = items)
                updateEmptyState( isEmpty = items.isEmpty())
            }
        }
    }
}
```

Figyeli a ViewModel items StateFlow-ját, és amikor változik, frissíti az adapter listáját és az üres állapot megjelenítését. A repeatOnLifecycle biztosítja, hogy csak akkor figyeljen, amikor az Activity STARTED állapotban van.

4.2 ShoppingViewModel.kt: üzleti logika kezelése

A ViewModel osztály kezeli az üzleti logikát és az adatforrásokkal való kommunikációt. A ViewModel túlél az Activity konfiguráció változásokon, így az adatok megmaradnak például képernyőforgatáskor.

Főbb tulajdonságok:

StateFlow-ok – Reaktív adatfolyamok:

```
3 Usages
private val _searchQuery = MutableStateFlow( value = "")
val searchQuery: StateFlow<String> = _searchQuery.asStateFlow()

3 Usages
private val _sortOption = MutableStateFlow( value = SortOption.DATE_DESC)
val sortOption: StateFlow<SortOption> = _sortOption.asStateFlow()

3 Usages
private val _filterCategory = MutableStateFlow<ItemCategory?>( value = null)
val filterCategory: StateFlow<ItemCategory?> = _filterCategory.asStateFlow()
```

StateFlow-okat használ a keresési lekérdezés, rendezési opció és kategória szűrő tárolásához. Ezek reaktív módon frissülnek, amikor változnak. A privát MutableStateFlow és a publikus StateFlow szétválasztása biztosítja az adatok védelmét.

items StateFlow – Szűrt és rendezett lista:

```
1 Usage
val items: StateFlow<List<ShoppingItem>> = combine(
    flow = repository.getAllItems(),
    flow2 = _searchQuery,
    flow3 = _sortOption,
    flow4 = _filterCategory
) { items, query, sort, category ->
    var filtered = items

    if (category != null) {
        filtered = filtered.filter { it.category == category }
    }

    if (query.isNotBlank()) {
        filtered = filtered.filter {
            it.name.contains( other = query, ignoreCase = true) ||
            it.notes.contains( other = query, ignoreCase = true)
        }
    }

    sort.sort( items = filtered)
}.stateIn(
    scope = viewModelScope,
    started = SharingStarted.WhileSubscribed( stopTimeoutMillis = 5000),
    initialValue = emptyList()
)
```

Kombinálja a repository adatait a keresési lekérdezéssel, rendezési opcióval és kategória szűrővel. Automatikusan újraszámolja a listát, amikor bármelyik bemenet változik. A WhileSubscribed(5000) biztosítja, hogy csak akkor fusson, amikor van aktív előfizető.

CRUD műveletek:

```
3 Usages
fun insertItem(item: ShoppingItem) {
    if (!item.isValid()) return
    viewModelScope.launch { repository.insertItem(item) }
}

1 Usage
fun updateItem(item: ShoppingItem) {
    if (!item.isValid()) return
    viewModelScope.launch { repository.updateItem(item) }
}

2 Usages
fun deleteItem(item: ShoppingItem) {
    viewModelScope.launch { repository.deleteItem(item) }
}
```

A ViewModel metódusok validálják az adatokat, majd coroutine-okban hívják meg a repository megfelelő metódusait. A viewModelScope automatikusan megszakítja a műveleteket, ha a ViewModel törlődik.

4.3 ShoppingRepository.kt (adatkezelési réteg)

A Repository absztrahálja az adatforrást és biztosítja az üzleti logika számára az adatok elérését.

Statisztikák számítása:

```
1 Usage
fun getStatistics(): Flow<ShoppingStatistics> {
    return dao.getAllFlow().map { items ->
        ShoppingStatistics(
            totalItems = items.size,
            boughtItems = items.count { it.isBought },
            pendingItems = items.count { !it.isBought },
            totalCost = items.filter { it.isBought }.sumOf { it.getTotalCost() },
            estimatedCost = items.sumOf { it.getTotalCost() }
        )
    }
}
```

A getStatistics() metódus a teljes elemek listájából számítja ki a statisztikákat: összes elem száma, vásárolt és függőben lévő elemek száma, valamint az elköltött és becsült összeg. A Flow automatikusan frissül, amikor az adatok változnak.

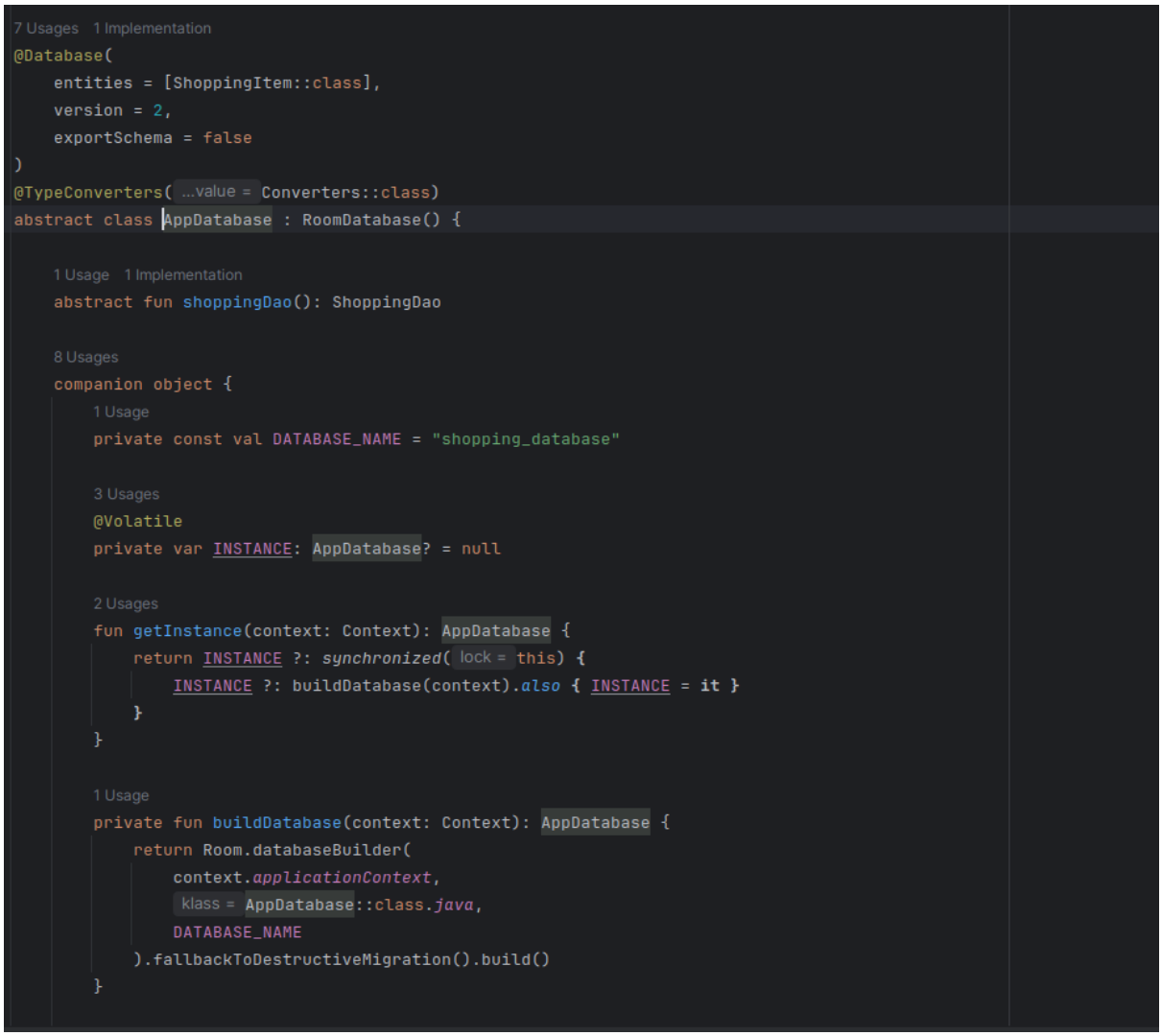
Lista megosztása:

```
1 Usage
suspend fun getShareableList(): String {
    val items = dao.getAll()
    if (items.isEmpty()) return "My Shopping List is empty"

    val builder = StringBuilder( str = "🏷 My Shopping List\n\n")
    val pending = items.filter { !it.isBought }
    val bought = items.filter { it.isBought }

    if (pending.isNotEmpty()) {
        builder.append("🛒 To Buy (${pending.size}):\n")
        pending.forEachIndexed { index, item ->
            builder.append("${index + 1}. ${item.name} (x${item.quantity}) - $%.2f\n".format( ...args = item.price))
        }
        builder.append("\n")
    }

    if (bought.isNotEmpty()) {
        builder.append("✅ Bought (${bought.size}):\n")
        bought.forEachIndexed { index, item ->
            builder.append("${index + 1}. ${item.name} (x${item.quantity}) - $%.2f\n".format( ...args = item.price))
        }
        builder.append("\n")
    }

    val totalCost = bought.sumOf { it.getTotalCost() }
    val estimatedTotal = items.sumOf { it.getTotalCost() }

    builder.append("💰 Spent: $%.2f\n".format( ...args = totalCost))
    builder.append("📊 Estimated Total: $%.2f".format( ...args = estimatedTotal))

    return builder.toString()
}
```

Formázott szöveget készít a lista megosztásához, amely tartalmazza a függőben lévő és vásárolt elemeket, valamint a költségstatisztikákat. A suspend függvény lehetővé teszi az aszinkron hívást.

4.4 AppDatabase.kt (Room adatbázis)

Room adatbázis singleton implementáció, amely biztosítja, hogy csak egy példány legyen az alkalmazás életciklusa során.

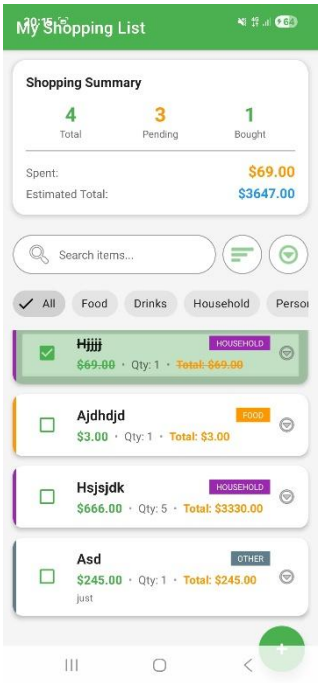


```
7 Usages  1 Implementation
@Database(
    entities = [ShoppingItem::class],
    version = 2,
    exportSchema = false
)
@TypeConverters( ...value = Converters::class)
abstract class AppDatabase : RoomDatabase() {

    1 Usage  1 Implementation
    abstract fun shoppingDao(): ShoppingDao

    8 Usages
    companion object {
        1 Usage
        private const val DATABASE_NAME = "shopping_database"

        3 Usages
        @Volatile
        private var INSTANCE: AppDatabase? = null

        2 Usages
        fun getInstance(context: Context): AppDatabase {
            return INSTANCE ?: synchronized( lock = this) {
                INSTANCE ?: buildDatabase(context).also { INSTANCE = it }
            }
        }

        1 Usage
        private fun buildDatabase(context: Context): AppDatabase {
            return Room.databaseBuilder(
                context.applicationContext,
                klass = AppDatabase::class.java,
                DATABASE_NAME
            ).fallbackToDestructiveMigration().build()
        }
```

Singleton mintát használ az adatbázis példány kezeléséhez. A getInstance() metódus biztosítja, hogy csak egy példány legyen az alkalmazás életciklusa során. A synchronized blokk biztosítja a thread-safety-t. A fallbackToDestructiveMigration() azt jelenti, hogy verzióváltáskor az adatbázis újraépül (fejlesztési célokra).

5. Screen-ek

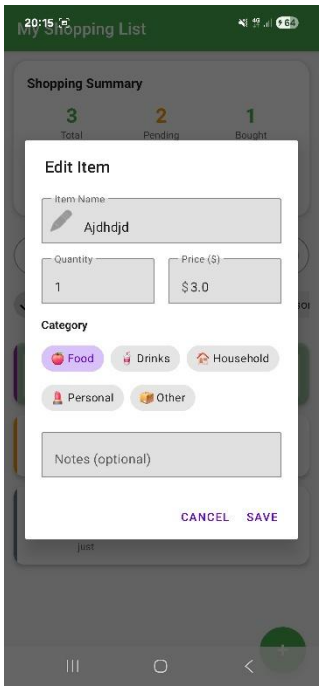
5.1 Főképernyő (Main Screen)



A főképernyő a bejelentkezett felhasználó fő nézete, ahol a saját bevásárlólistaelemei jelennek meg. Felül összefoglaló statisztikák láthatók (összes elem, függőben lévő, vásárolt elemek száma, elköltött és becsült összeg), alatta pedig keresőmező, kategória szűrő chipek és a lista elemek. A felhasználó kereshet, szűrhet kategória szerint, valamint rendezheti a tartalmakat. A tartalmak kártyák formájában jelennek meg, amelyeken látszik a kategória színes jelzője, alapadatok és az árinformáció. A FloatingActionButton-nal új elemek adhatók hozzá, a kártyákra kattintva pedig szerkesztés és törlés végezhető.

A főképernyő Material Design elveket követ: MaterialCardView a statisztikáknak, MaterialToolbar az app bar-hoz, TextInputLayout a keresőmezőhöz, ChipGroup a kategória szűrőkhöz. Az üres állapot esetén egy üzenet jelenik meg, amely arra ösztönzi a felhasználót, hogy adjon hozzá első elemet.

5.2 Hozzáadás/Szerkesztés dialógus



A hozzáadás/szerkesztés dialógus részletes űrlapot tartalmaz az elem adataihoz: név, mennyiség, ár, kategória, jegyzetek. A felhasználó minden mezőt módosíthat, majd a „Save” gombbal mentheti az adatokat. A dialógus Material Design TextInputLayout komponenseket használ outline stílussal, amelyek modern és felhasználóbarát megjelenést biztosítanak.

Az űrlap beépített validációs logikával rendelkezik: ellenőrzi a kötelező mezők kitöltését (név), a mennyiség és ár pozitív értékét, valamint hibás vagy hiányos adatok esetén vizuális visszajelzést ad a felhasználónak Snackbar formájában. Ilyen esetekben a rendszer nem engedi tovább a mentést, amíg minden mező helyesen nem kerül kitöltésre.

Ez a megoldás biztosítja, hogy az elemek csak érvényes és konzisztens adatokkal kerüljenek mentésre, javítva ezzel a felhasználói élményt és a rendszer megbízhatóságát.

5.3 Statisztikák megjelenítése

A statisztikák kártya a főképernyő tetején jelenik meg, amely összefoglaló információkat tartalmaz a bevásárlólistáról. A kártya három fő metrikát mutat: összes elem száma, függőben lévő elemek száma és vásárolt elemek száma. Alatta két költség információ látható: az elköltött összeg (csak a vásárolt elemekből) és a becsült összeg (összes elem).

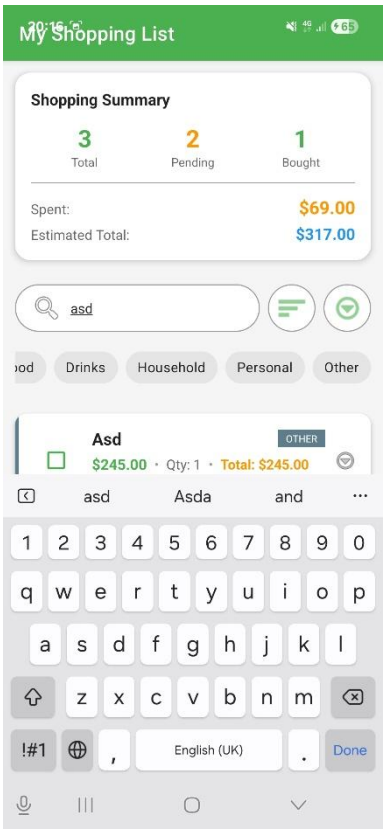
A statisztikák valós időben frissülnek, amikor az elemek változnak. A ViewModel statistics StateFlow-ja automatikusan újraszámolja az értékeket, amikor új elemeket adnak hozzá, törölnek vagy módosítanak. A számok színes megjelenítése (primary, warning, success színek) segíti a felhasználót a gyors információfeldolgozásban.

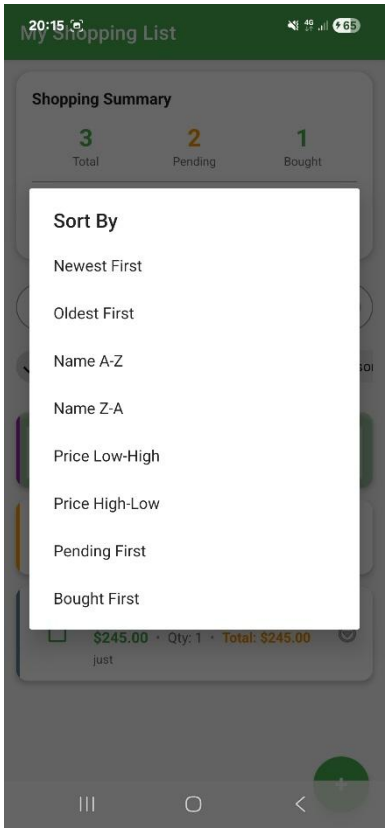
5.4 Keresés és szűrés funkciók

A keresési funkció lehetővé teszi a felhasználó számára, hogy név vagy jegyzetek alapján keressen az elemek között. A keresőmező fölött található, és valós időben szűri a listát ahogy a felhasználó gépel. A keresés case-insensitive, így nem számít a kis- és nagybetűk különbsége.

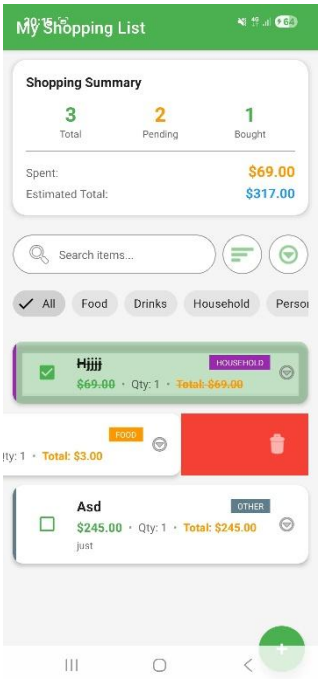
A kategória szűrés chip gombokkal történik, amelyek a keresőmező alatt helyezkednek el. A felhasználó kiválaszthatja az "All" opciót (minden kategória), vagy egy konkrét kategóriát (Food, Drinks, Household, Personal, Other). A kiválasztott chip vizuálisan kiemelkedik, és a lista azonnal frissül csak az adott kategóriájú elemekkel.

A rendezési funkció egy dialóguson keresztül érhető el, ahol a felhasználó választhat különböző rendezési opciók közül: dátum szerint (legújabb/legrégebbi), értékelés szerint vagy cím szerint (A-Z/Z-A). A kiválasztott rendezési opció azonnal alkalmazkodik a listára.





5.5 Swipe-to-delete funkció



A swipe-to-delete funkció lehetővé teszi a felhasználó számára, hogy balra húzással töröljön egy elemet a listából. Ez egy intuitív és gyors módszer az elemek eltávolítására anélkül, hogy meg kellene nyitni a szerkesztési menüt.

Amikor egy elem törlődik swipe művelettel, egy Snackbar jelenik meg az alsó részén, amely tájékoztatja a felhasználót a törlésről, és egy "UNDO" gombot tartalmaz. Ha a felhasználó rákattint az "UNDO" gombra, az elem visszaállításra kerül. Ez biztosítja, hogy a véletlen törlések könnyen visszafordíthatók legyenek.

A swipe animáció sima és természetes, amely javítja a felhasználói élményt. A SwipeToDeleteCallback osztály kezeli a swipe észlelését és a törlési műveletet.

5.6 Lista megosztása

A lista megosztása funkció lehetővé teszi a felhasználó számára, hogy formázott szöveg formában megossza a bevásárlólistát más alkalmazásokkal (pl. üzenetküldő, email). A funkció a menü gombból érhető el, amely a keresőmező mellett található.

A megosztott lista tartalmazza a függőben lévő és vásárolt elemeket külön csoportokban, minden elemmel együtt a mennyiségével és árával. A lista végén látható az elköltött összeg és a becsült összeg is. Ez lehetővé teszi, hogy a felhasználó könnyen megossza a listát családtagokkal vagy barátokkal.

A megosztás Android Intent rendszerét használja, amely lehetővé teszi, hogy a felhasználó válasszon a rendelkezésre álló megosztási opciók közül (pl. WhatsApp, Email, Messaging).