Jegyzőkönyv

Bucket Buddy

Benkő Péter

BEVEZETÉS

A dokumentáció a Bucket Buddy Android alkalmazás fejlesztését, felépítését és működését mutatja be.
Az alkalmazás célja, hogy a felhasználók személyes céljaikat (bucket list / bakancslista elemeket) rögzíthessék, nyomon követhessék, majd teljesítés után archiválhassák.

A projekt modern Android fejlesztési eszközökkel készült, Kotlin programozási nyelven, Jetpack Compose felhasználói felülettel, valamint Room ORM alapú SQLite adatbázissal. A fejlesztés során kiemelt szempont volt az átlátható kódstruktúra, a rétegzett architektúra és a felhasználóbarát működés.

A PROJEKT CÉLJA ÉS KÖVETELMÉNYEI

A projekt fő célja egy egyszerű, mégis jól strukturált CRUD (Create, Read, Update, Delete) alkalmazás létrehozása Android platformra. A forráskód elérhető ezen a linken.

Funkcionális követelmények:

- Új bakancslista elem létrehozása
- Meglévő elem szerkesztése
- Elem törlése
- Elem teljesítetté jelölése
- Aktív és teljesített elemek külön nézetben való megjelenítése
- Teljesítés dátumának rögzítése

Nem funkcionális követelmények:

- Modern felhasználói felület
- Sötét színvilág
- Gyors adatkezelés
- Karbantartható kódbázis
- Hibás bevitel megakadályozása

HASZNÁLT TECHNOLÓGIÁK

A projekt során az alábbi technológiák kerültek felhasználásra:

- Kotlin: elsődleges programozási nyelv
- Jetpack Compose: deklaratív UI fejlesztés
- Material 3: modern dizájn komponensek
- Room ORM: SQLite adatbázis kezelés
- MVVM architektúra
- Coroutines: aszinkron műveletek
- Gradle (KTS): build rendszer

AZ ALKALMAZÁS MŰKÖDÉSE FELHASZNÁLÓI SZEMSZÖGBŐL

Az alkalmazás indítását követően a felhasználó azonnal a főképernyőre jut, amely az aktív, még nem teljesített bakancslista elemeket jeleníti meg.

Amennyiben a felhasználó még nem hozott létre egyetlen bakancslista elemet sem, az alkalmazás egy üres állapotot jelenít meg. Ebben az állapotban egy rövid szöveges tájékoztatás segíti a felhasználót abban, hogy megértse az alkalmazás működését és felismerje az első lépést.

Új elem hozzáadása a képernyő jobb alsó sarkában elhelyezkedő lebegő műveleti gomb segítségével történik. A gomb megnyomására egy felugró párbeszédablak jelenik meg, amelyben a felhasználó megadhatja az új elem adatait. A felhasználó köteles megadni az elem címét és a cél dátumát, míg a leírás és a kategória kiegészítő információként szolgálnak. A prioritás mező numerikus értéket vár 1-től 5-ig, amely segíti a cél fontosságának meghatározását.

A párbeszédablak valós időben ellenőrzi a bevitt adatokat. Amennyiben valamely kötelező mező hiányzik, vagy a dátum formátuma nem megfelelő, az alkalmazás nem engedi az adat mentését. Ez a megoldás megakadályozza a hibás adatok rögzítését, és javítja a felhasználói élményt.

A létrehozott bakancslista elemek kártya formájában jelennek meg a főképernyőn. Minden kártya tartalmazza az elem legfontosabb információit: címet, leírást, kategóriát, prioritást és cél dátumot. A kártyák alsó részén műveleti ikonok találhatók, amelyek lehetővé teszik az elem szerkesztését, törlését, illetve teljesítetté jelölését.

Egy elem teljesítetté jelölése egyetlen művelettel elvégezhető. A felhasználó a megfelelő ikonra kattintva az elemet befejezettnek jelöli, amelynek hatására az adott elem automatikusan eltűnik az aktív listából. Ezzel egy időben az alkalmazás rögzíti a teljesítés dátumát, amely később megjelenik a teljesített elemek nézetben.

A főképernyő felső részén található állapotváltó vezérlő segítségével a felhasználó bármikor átválthat a teljesített elemek nézetére. Ebben a nézetben kizárólag a már lezárt célok jelennek meg.

A teljesített elemek listájában a felhasználó továbbra is megtekintheti az elemek részleteit, valamint lehetősége van azok törlésére. A státusz visszaállítása azonban nem engedélyezett, így az alkalmazás biztosítja a célok lezártságának véglegességét.

FELHASZNÁLÓI FELÜLET ÉS DIZÁJN

Az alkalmazás felhasználói felülete a Jetpack Compose deklaratív UI keretrendszer segítségével készült, amely lehetővé teszi az átlátható, moduláris és könnyen karbantartható felület kialakítását.

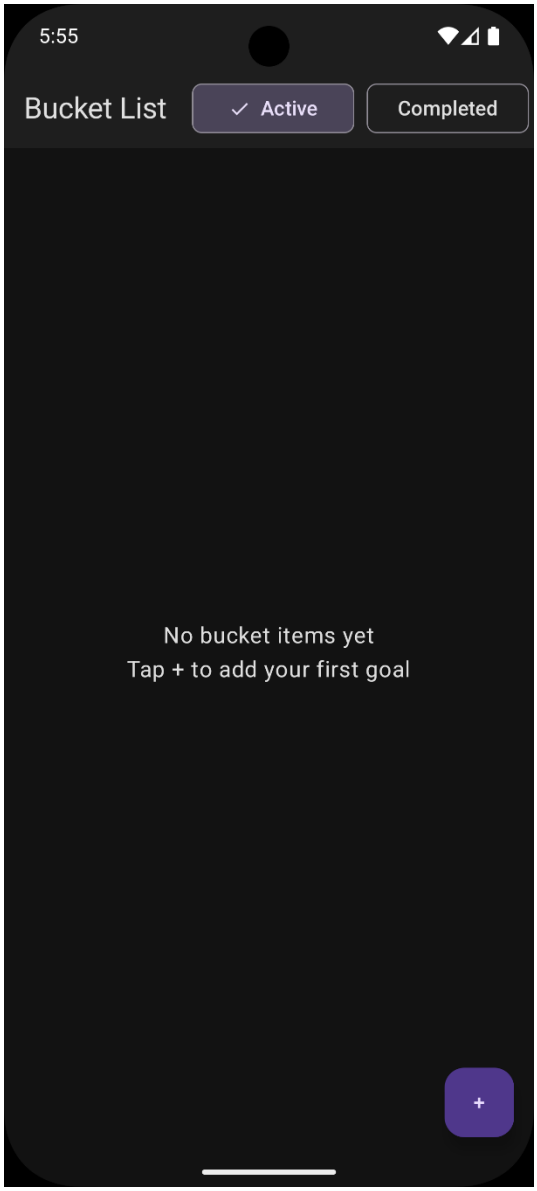
A bakancslista elemek megjelenítése kártya alapú elrendezéssel történik. Minden kártya egy önálló egységet képez, amely tartalmazza az adott cél legfontosabb adatait. A címsor nagyobb betűmérettel és hangsúlyosabb stílussal jelenik meg, míg a leírás és a kiegészítő adatok kisebb, visszafogottabb tipográfiát kapnak.

A kártyák alsó részén elhelyezett ikon alapú vezérlők biztosítják az egyes elemekhez tartozó műveletek gyors elérését.

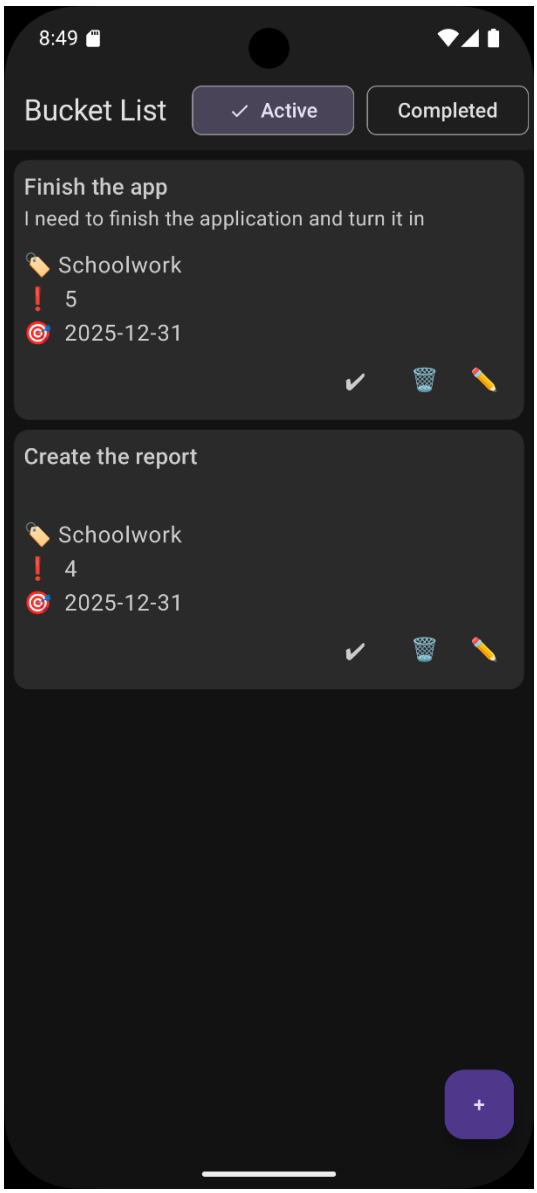
Az új elemek hozzáadását szolgáló lebegő műveleti gomb a képernyő jobb alsó sarkában található. A gomb megnyomására megjelenő párbeszédablak a képernyő többi részétől vizuálisan elkülönül, ezzel a felhasználó figyelmét a bevitelre összpontosítja.

A főképernyő elemei:

- Felső alkalmazássáv címmel
- Állapotváltó (Aktív / Teljesített)
- Kártya alapú lista
- Lebegő műveleti gomb

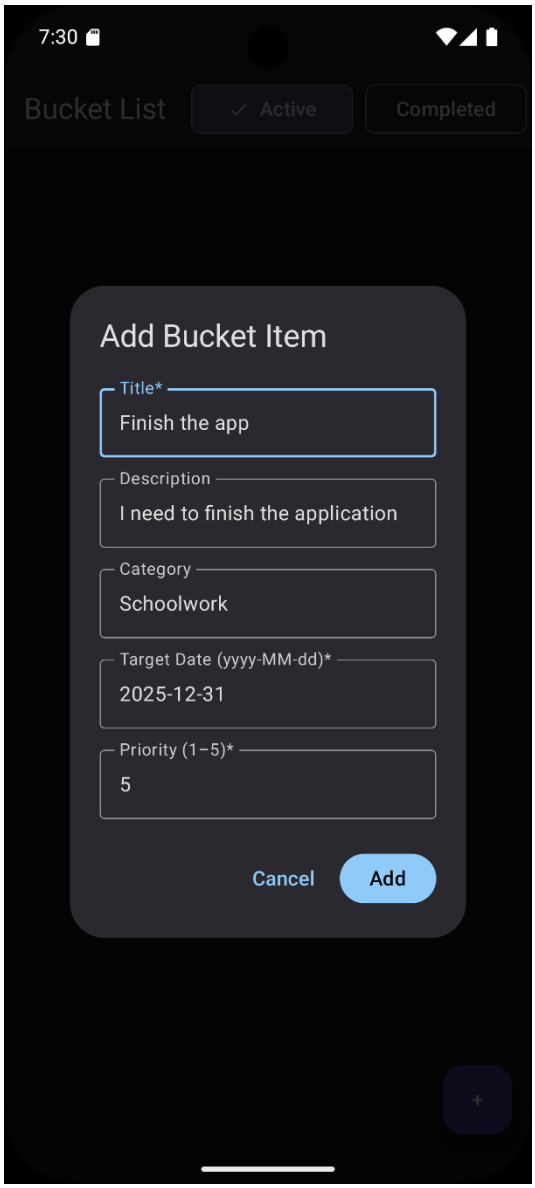


(Elemek nélkül)

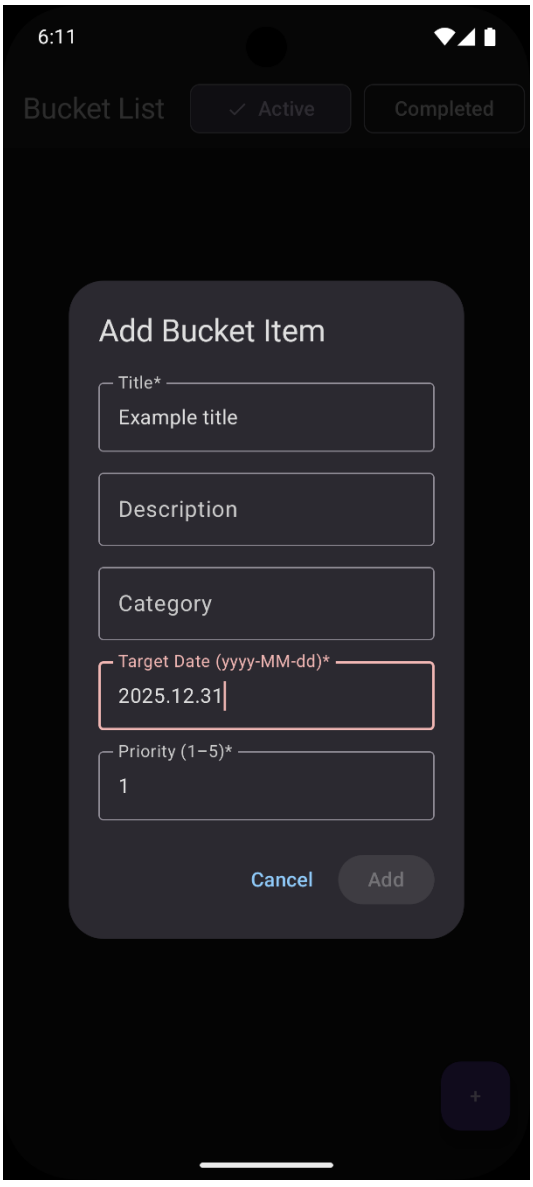


(Hozzáadott elemekkel)

A hozzáadás és szerkesztés során használt párbeszédablak hibás vagy hiányzó adatok esetén azonnali visszajelzést ad.

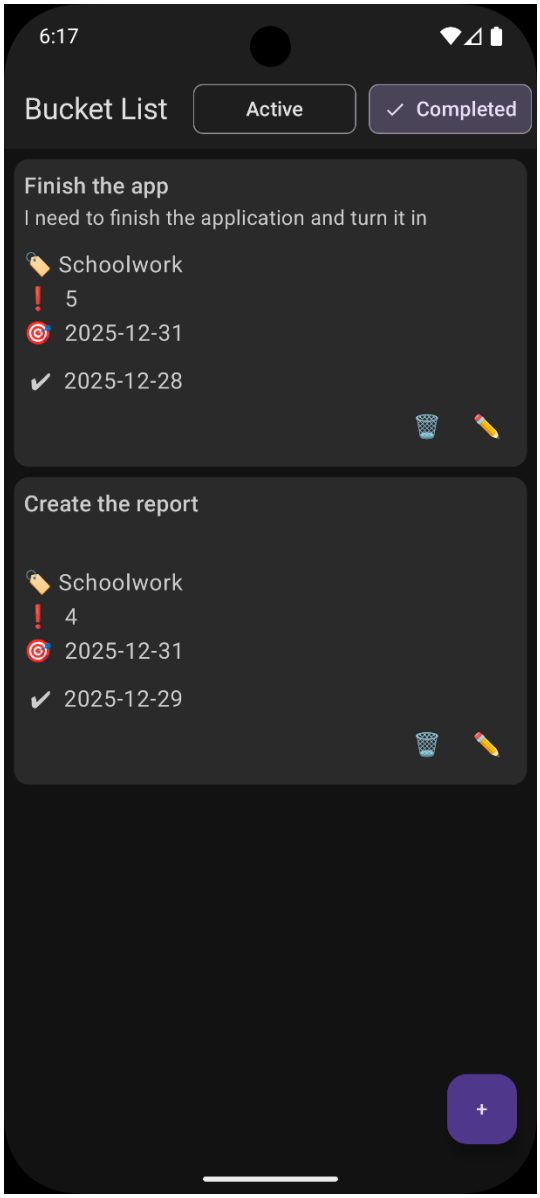


(Hozzáadás/szerkesztés ablak)



(Hiba kiemelése)

A teljesített elemek nézetben a kártyákon ugyanaz az ikon (✓) jelzi az adott cél teljesítésének időpontját. Teljesített célt már csak szerkeszteni és törölni tudunk.



(Teljesített célok)

PROJEKT STRUKTÚRA

A projekt logikailag elkülönített csomagokra van bontva:

- data
    - BucketItem (adatmodellek)
    - BucketItemDao (adatbázis műveletek)
    - BucketDatabase (Room adatbázis)

- ui
    - components (újrahasználható UI elemek)
        - StatusToggle
    - screens (képernyők)
        - AddItemDialog
        - MainScreen
    - theme (színek, tipográfia)
        - Theme

- viewmodel
    - BucketViewModel

A Bucket Buddy alkalmazás forráskódja logikailag elkülönített csomagokra (package-ekre) van bontva annak érdekében, hogy az alkalmazás jól áttekinthető, könnyen karbantartható és később bővíthető maradjon.

Entitás (BucketItem)

Az alkalmazás központi adatmodellje a BucketItem osztály, amely egy adatbázis táblát reprezentál.

```
@Entity(tableName = "bucket_items")
data class BucketItem(
    @PrimaryKey(autoGenerate = true)
    val id: Long = 0,
    val title: String,
    val description: String,
    val category: String,
    val targetDate: String,
    val priority: Int,
    val status: Boolean = false,
    val completedAt: String? = null
)
```

Ez az osztály határozza meg az adatbázis tábla szerkezetét. A completedAt mező opcionális, amely lehetővé teszi, hogy csak a teljesített elemek esetén kerüljön kitöltésre.

A @Entity annotáció jelzi a Room-nak, hogy ez az osztály egy entitást reprezentál. Az elemek azonosítója automatikusan generálódik.

DAO (BucketItemDao)

A DAO interfész tartalmazza az adatbázis-műveletekhez szükséges metódusokat. Ezek biztosítják az adatok beszúrását, frissítését, törlését és lekérdezését.

Itt látható egy példa a még nem teljesített (status = 0) célok lekérdezésére:

```
@Query("SELECT * FROM bucket_items WHERE status = 0 ORDER BY priority
DESC")
suspend fun getActiveItems(): List<BucketItem>
```

Az adatok prioritás szerinti csökkenő sorrendben érkeznek.

Itt pedig a cél teljesítése esetén lefutó query látható:

```
@Query("UPDATE bucket_items SET status = 1, completedAt = :completedAt
WHERE id = :id")
suspend fun markCompleted(id: Long, completedAt: String)
```

A status mező értéke 1-re változik, ami a teljesítettséget jelöli, valamint bekerül a teljesítés dátuma.

A hozzáadás, általános módosítás és törlés művelethez a ROOM biztosít külön annotációkat, így ezeknél nem szükséges kézzel megírni az SQL parancsot.

Példa:

```
@Insert
suspend fun insert(item: BucketItem)
```

Adatbázis és migráció

Az adatbázis definíciója egy absztrakt osztályban történik, amely örökli a RoomDatabase osztályt. Az alkalmazás verziókezelt adatbázist használ, így az adatstruktúra változásai migrációval történnek, ami biztosítja, hogy az alkalmazás frissítésekor a meglévő adatok ne vesszenek el.

Ennek működését ki is próbáltam, mert eredetileg a teljesítés dátuma nem került mentésre, ez későbbi ötlet volt, amikor már voltak teszt adatok az adatbázisban, ezért látható 2-es verziószám.

```
@Database(
    entities = [BucketItem::class],
    version = 2,
    exportSchema = false
)
abstract class BucketDatabase : RoomDatabase() {

        abstract fun bucketItemDao(): BucketItemDao
        …
}
```

UI

Téma

Az alkalmazás kinézete sötét megjelenési tematikát követ, ami legfőképp személyes preferencia, de emellett kíméli a szemet, telefon technológiától függően csökkentheti az energiafogyasztást. Főszíneknek a lilát és kéket választottam.

Az alkalmazás színpalettája

```
private val DarkColorScheme = darkColorScheme(
    primary = Color(0xFF90CAF9),
    onPrimary = Color.Black,

    secondary = Color(0xFFB39DDB),
    onSecondary = Color.Black,

    background = Color(0xFF121212),
    onBackground = Color(0xFFE0E0E0),

    surface = Color(0xFF1E1E1E),
    onSurface = Color(0xFFE0E0E0),

    surfaceVariant = Color(0xFF2A2A2A),
    onSurfaceVariant = Color(0xFFCCCCCC)
)
```

A színeket többnyire hex formátumban határoztam meg a maradék esetben az alapértelmezett feketét használtam.

Megjelenítés

A főképernyőért felelős composable függvény egy külön fájlban található, amely a lista megjelenítését, az állapotváltást és a lebegő műveleti gomb kezelését végzi.

```
var showingCompleted by remember { mutableStateOf(false) }
```

Ez a változó határozza meg, hogy az alkalmazás éppen az aktív vagy a teljesített elemeket jelenítse meg. A változó értékének módosítása automatikusan újra rajzolja a felhasználói felületet, amely a Compose reaktív működésének egyik alapelve.

```
@Composable
fun StatusToggle(
    showingCompleted: Boolean,
    onToggle: (Boolean) -> Unit
) {
    SingleChoiceSegmentedButtonRow {
        SegmentedButton(
            selected = !showingCompleted,          // Active button
            onClick = { onToggle(false) },
            shape = RoundedCornerShape(8.dp)
        ) {
            Text("Active")
        }
        …
}
```

ViewModel

A ViewModel az alkalmazás üzleti logikáját tartalmazza. Feladata az adatok lekérése az adatbázisból, valamint az állapot kezelése a felhasználói felület számára.

A ViewModel korutinokat (coroutine) használ a háttérben futó műveletekhez, így az adatbázis-műveletek nem blokkolják a felhasználói felületet.

Példa egy elem törlésére:

```
fun delete(item: BucketItem) {
    viewModelScope.launch {
        repository.delete(item)
        refresh()
    }
}
```

Ez a metódus törli az elemet, majd újratölti az aktív elemek listáját.

TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

- Dátumválasztó komponens használata
- Keresési és szűrési funkció
- Felhő alapú szinkronizáció
- Animációk a listaelemekhez
- Megerősítés elem törlése és teljesítése előtt

Miskolc, 2025. 12. 29.