Mobilprogramozás alapjai

BEADANDÓ FELADAT JEGYZŐKÖNYV

Tóth Mátyás | OQH5NH

1. Tartalomjegyzék

1. Tartalomjegyzék ---- 1
2. Bevezetés ---- 2
3. Feladat leírása ---- 3
4. Projekt struktúra ---- 3

EntryPoint (Belépési Pont) ---- 4

Data Layer (Adatréteg) ---- 5

UI Layer (Felhasználói felület réteg) ---- 5

Theme (Téma) ---- 6

5. Felhasználói felület ---- 6

Főképernyő (CarListScreen) ---- 7

Autó kártya (CarListItem) ---- 9

Új autó hozzáadása ---- 10

Autó szerkesztése ---- 12

Autó törlése ---- 13

Többszörös kiválasztás (Multi-Select Mode) ---- 14

Tömeges szerkesztés ---- 16

Tömeges törlés ---- 17

Összes autó törlése ---- 17

Szűrés és rendezés ---- 18

Üres állapot ---- 19

Visszajelzések ---- 19

6. Kód magyarázat ---- 20

MVVM Architektúra ---- 21

CarItem Entity (Adatmodell) ---- 21

CarItemDao (Adatbázis hozzáférés) ---- 22

AppDatabase (Adatbázis singleton) ---- 22

CarRepository (Repository pattern) ---- 23

CarViewModel (Állapotkezelés) ---------- 24
CarListScreen (UI komponens) ---------- 26
CarDialog (Hozzáadás/Szerkesztés dialógus) ---------- 27
Többszörös kiválasztás logika ---------- 28
7. Adatbázis struktúra ---------- 29
Táblaséma ---------- 29
Mezők leírása ---------- 29
Lekérdezések ---------- 30
Mintaadatok ---------- 30

2. Bevezetés

Ez egy Autóbérlő Kezelő Rendszer Android alkalmazás, Kotlin nyelven, Jetpack Compose UI-val és Room adatbázissal. A projekt egyetemi beadandó, amely teljes CRUD műveleteket valósít meg: autók hozzáadása, listázása, szerkesztése és törlése, validációval, tömeges műveletekkel (többszörös kiválasztás, tömeges törlés/szerkesztés), valamint keresés, szűrés és rendezés funkciókkal. Az alkalmazás MVVM architektúrát követ, OOP elveket alkalmaz, SQLite adatbázist használ Room-mal, és Material Design 3 felületet tartalmaz sötétkék színsémával. A jegyzőkönyv dokumentálja a projekt struktúráját, a felhasználói felületet, a kód magyarázatát és az adatbázis sémát, részletes kódpéldákkal és magyarázatokkal, hogy bemutassa a megvalósítás minden aspektusát.

3. Feladat leírása

A feladat egy Android alkalmazás elkészítése volt, amelynek tartalmaznia kellett:

FŐ KÖVETELMÉNYEK:

- **CRUD műveletek**: Autók hozzáadása, listázása, szerkesztése, törlése
- **OOP elvek**: Megfelelő objektum-orientált programozás, modellek, osztályok
- **SQLite adatbázis**: Room adatbázis integráció, adatok lekérdezése és írása
- **Adatvalidáció**: Bemeneti adatok ellenőrzése autók és bérlések hozzáadásakor
- **Tömeges műveletek**: Több autó törlése/szerkesztése egyszerre

A választott témakör egy kitalált autóbérlő kezelő Android alkalmazás, ahol az autóbérlő tulajdonosa tudja kezelni a flottáját, nyomon követheti éppen melyik autó szabad és különböző szűrőkkel tudja listázni az autóit.

4. Projekt struktúra

A projekt fájlstruktúrája a következő:

```
CarRental/
├── app/
│   ├── build.gradle.kts              # App szintű build konfiguráció
│   └── src/main/
│       ├── java/com/example/carrental/
│       │   ├── MainActivity.kt       # Fő Activity, alkalmazás belépési pontja
│       │   ├── data/
│       │   │   ├── model/
│       │   │   │   └── CarItem.kt    # Autó entitás modell validációval
│       │   │   ├── dao/
│       │   │   │   └── CarItemDao.kt # Adatbázis hozzáférési objektum (DAO)
│       │   │   ├── database/
│       │   │   │   └── AppDatabase.kt # Room adatbázis singleton
│       │   │   └── repository/
│       │   │       └── CarRepository.kt # Repository pattern implementáció
│       │   └── ui/
│       │       ├── viewmodel/
│       │       │   └── CarViewModel.kt # ViewModel állapotkezeléssel
│       │       ├── screens/
│       │       │   └── CarListScreen.kt # Fő képernyő komponens
│       │       ├── components/
│       │       │   ├── CarListItem.kt    # Autó lista elem komponens
│       │       │   ├── CarDialog.kt      # Hozzáadás/szerkesztés dialógus
│       │       │   ├── BatchEditDialog.kt # Tömeges szerkesztés dialógus
│       │       │   └── EmptyState.kt     # Üres állapot komponens
│       │       └── theme/
│       │           ├── Color.kt          # Színpaletta definíciók
│       │           ├── Theme.kt          # Material3 téma konfiguráció
│       │           └── Type.kt           # Tipográfia beállítások
│       └── AndroidManifest.xml
├── build.gradle.kts                   # Projekt szintű build konfiguráció
└── gradle/
    └── libs.versions.toml             # Függőségek verziókezelése
```

EntryPoint (Belépési Pont)

MainActivity.kt:

- Az alkalmazás belépési pontja
- Inicializálja a Jetpack Compose UI-t
- Betölti a CarListScreen komponenst

Data Layer (Adatréteg)

CarItem.kt:

- Room entitás, amely az autó adatait reprezentálja
- Validációs függvények minden mezőre (brand, model, year, color, licensePlate, dailyRate)
- ValidationResult data class a validációs eredményekhez

CarItemDao.kt:

- Adatbázis hozzáférési objektum interface
- 15+ lekérdezési metódus (getAllCars, getAvailableCars, getRentedCars, stb.)
- Flow-alapú reaktív lekérdezések
- Tömeges műveletek támogatása

AppDatabase.kt:

- Room adatbázis singleton implementáció
- Verziókezelés (version 1)
- Automatikus mintaadatok betöltése első indításkor (5 autó)

CarRepository.kt:

- Repository pattern implementáció
- Elválasztja a ViewModel-t az adatbázis rétegtől
- Coroutine context kezelés (Dispatchers.IO)

UI Layer (Felhasználói felület réteg)

CarViewModel.kt:

- AndroidViewModel leszármazott
- StateFlow-alapú állapotkezelés
- CRUD műveletek implementációja
- Keresés, szűrés, rendezés logika

- Többszörös kiválasztás (multi-select) kezelése
- Statisztikák számítása

CarListScreen.kt:
- Fő képernyő komponens
- Scaffold struktúra TopAppBar-ral és FAB-bal
- LazyColumn autók listázásához
- Dialógusok kezelése (hozzáadás, szerkesztés, törlés)
- Keresőmező, szűrők, rendezés UI

CarListItem.kt:
- Egyedi autó kártya komponens
- Autó adatainak megjelenítése
- Kiválasztási checkbox multi-select módban
- Szerkesztés/Törlés gombok
- Bérlési státusz váltó chip

CarDialog.kt:
- Hozzáadás/Szerkesztés dialógus
- Minden mező validációval
- Hibaüzenetek megjelenítése
- Mentés/Mégse gombok

BatchEditDialog.kt:
- Tömeges szerkesztés dialógus
- Opcionális mezők (napi díj, szín, bérlési státusz)
- Validáció csak a módosított mezőkre

EmptyState.kt:
- Üres állapot komponens
- Megjelenik amikor nincs autó
- "Add hozzá az első autódat" gomb

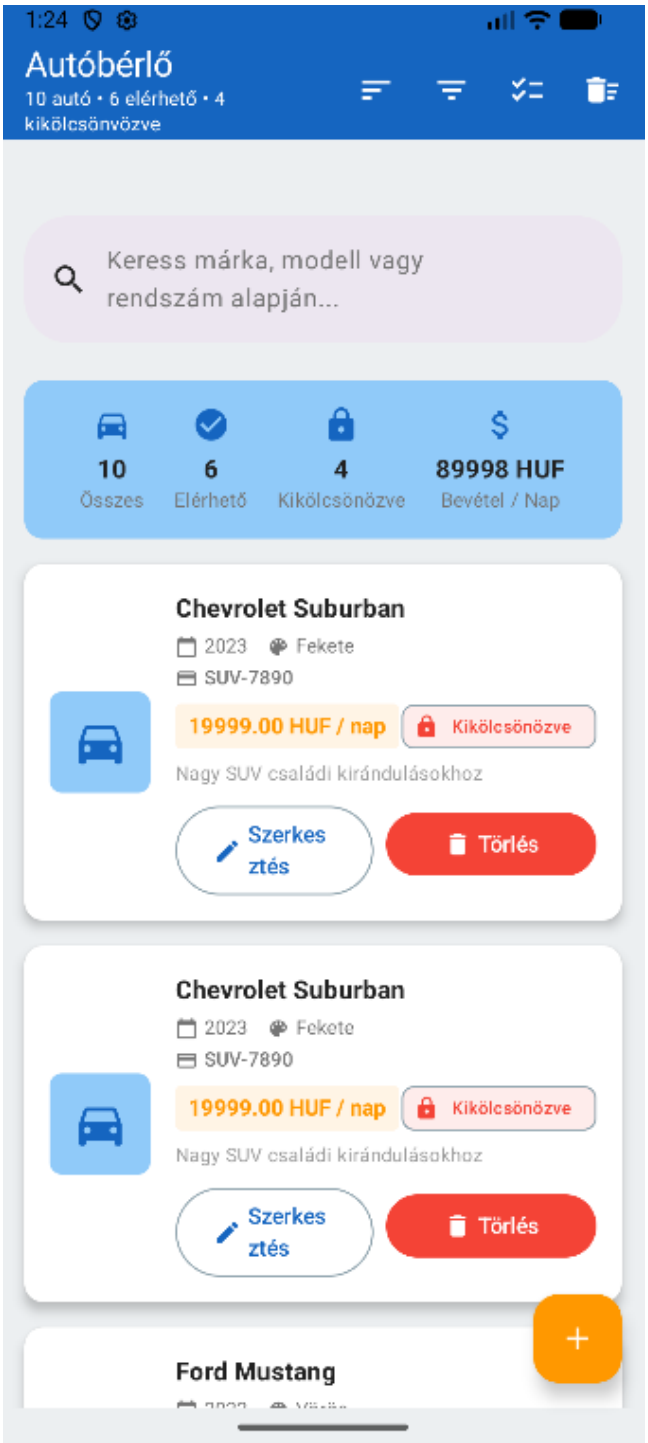
Theme (Téma)

- **Color.kt**: Sötétkék színpaletta definíciók
- **Theme.kt**: Material3 téma konfiguráció dark/light módokkal

5. Felhasználói felület

Főképernyő (CarListScreen)

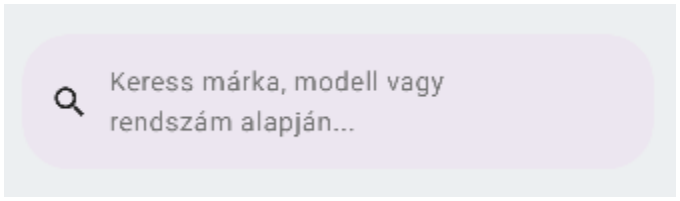
Az alkalmazás egyetlen főképernyőből áll, amely tartalmazza az összes funkciót:

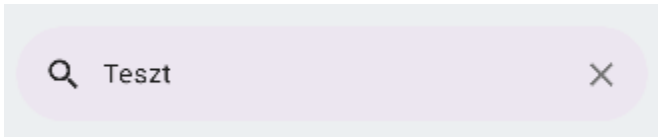


Főbb elemek:

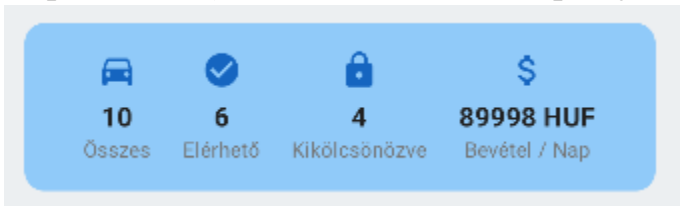
1. **Top App Bar (Felső navigációs sáv)**
   - Cím: "Autóbérlő"
   - Statisztikák: "X autó • Y elérhető • Z kikölcsönvözve"
   - Akció gombok: Rendezés, Szűrés, Többszörös kiválasztás, Összes törlése

2. **Keresőmező**
   - Valós idejű keresés márka, modell vagy rendszám alapján
   - Törlés gomb a keresés megszüntetéséhez



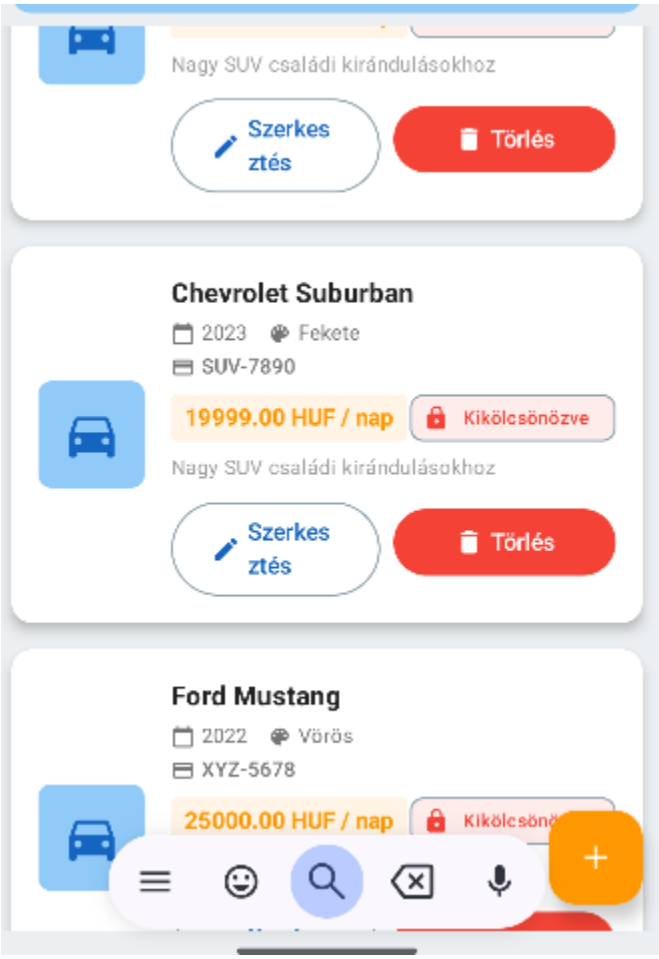


3. **Statisztikák kártya**
   - Összes autó száma
   - Elérhető autók száma
   - Kikölcsönzött autók száma
   - Napi bevétel (kikölcsönzött autók napi díjainak összege)



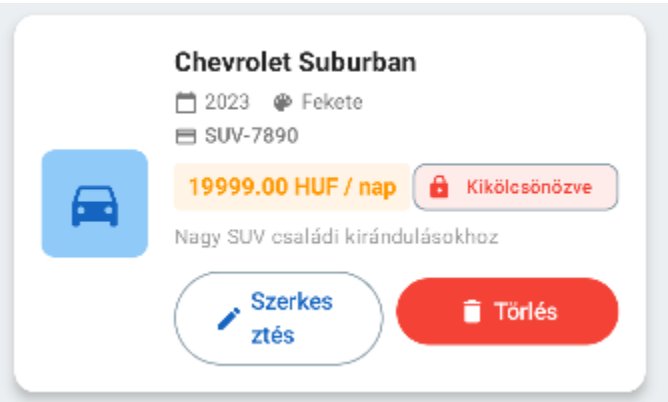
4. **Autók listája**
   - LazyColumn-ban scrollozható lista
   - Minden autó egy kártyában jelenik meg
   - Autó ikon, márka-modell, évjárat, szín, rendszám, napi díj, bérlési státusz

- Szerkesztés és Törlés gombok



Autó kártya (CarListItem)

Minden autó egy Material3 kártyában jelenik meg:



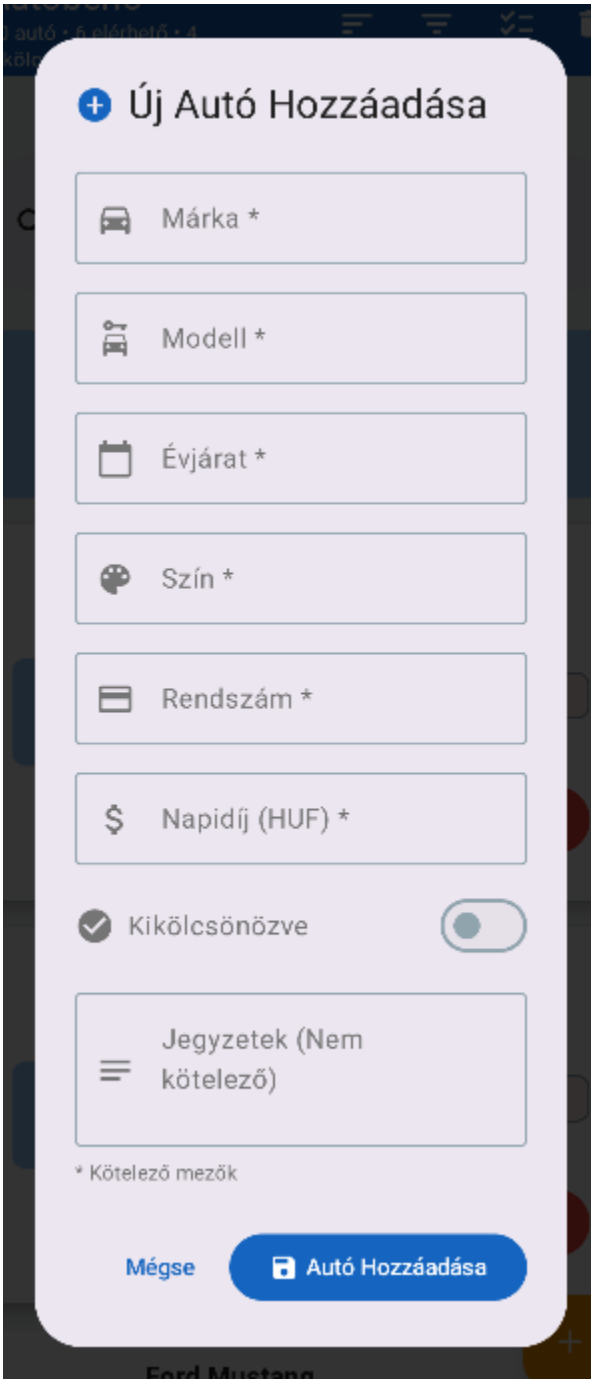
Tartalom:

- **Autó ikon**: Kék háttérrel, autó ikonnal
- **Márka és modell**: Nagy, félkövér szöveg (pl. "Toyota Camry")
- **Évjárat és szín**: Ikonokkal, kisebb szöveg

- **Rendszám**: Külön sorban, félkövér betűtípussal
- **Napi díj**: Narancssárga badge-ben (pl. "10000.00 HUF / nap")
- **Bérlési státusz chip**:
    - Zöld "Elérhető" chip, ha nincs kikölcsönzve
    - Piros "Kikölcsönzött" chip, ha ki van kölcsönözve
    - Kattintható a státusz váltásához
- **Szerkesztés gomb**: Kék outlined gomb
- **Törlés gomb**: Piros gomb
- **Megjegyzések**: Opcionálisan megjelenik, ha van

Új autó hozzáadása

A Floating Action Button (FAB) megnyomásakor megnyílik a hozzáadás dialógus:



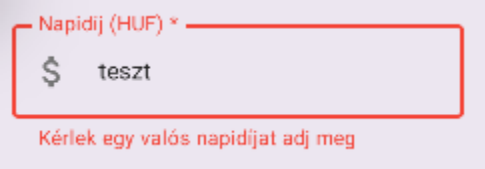
Mezők:

- **Márka***: Szöveges mező (2-50 karakter)
- **Modell***: Szöveges mező (1-50 karakter)
- **Évjárat***: Szám mező (1900-jelenlegi év+1)
- **Szín***: Szöveges mező (2-30 karakter)

- **Rendszám***: Szöveges mező (5-10 alfanumerikus karakter)
- **Napi díj (HUF)***: Tizedes szám mező
- **Jelenleg kikölcsönzött**: Kapcsoló (switch)
- **Megjegyzések**: Többsoros szöveges mező (opcionális)

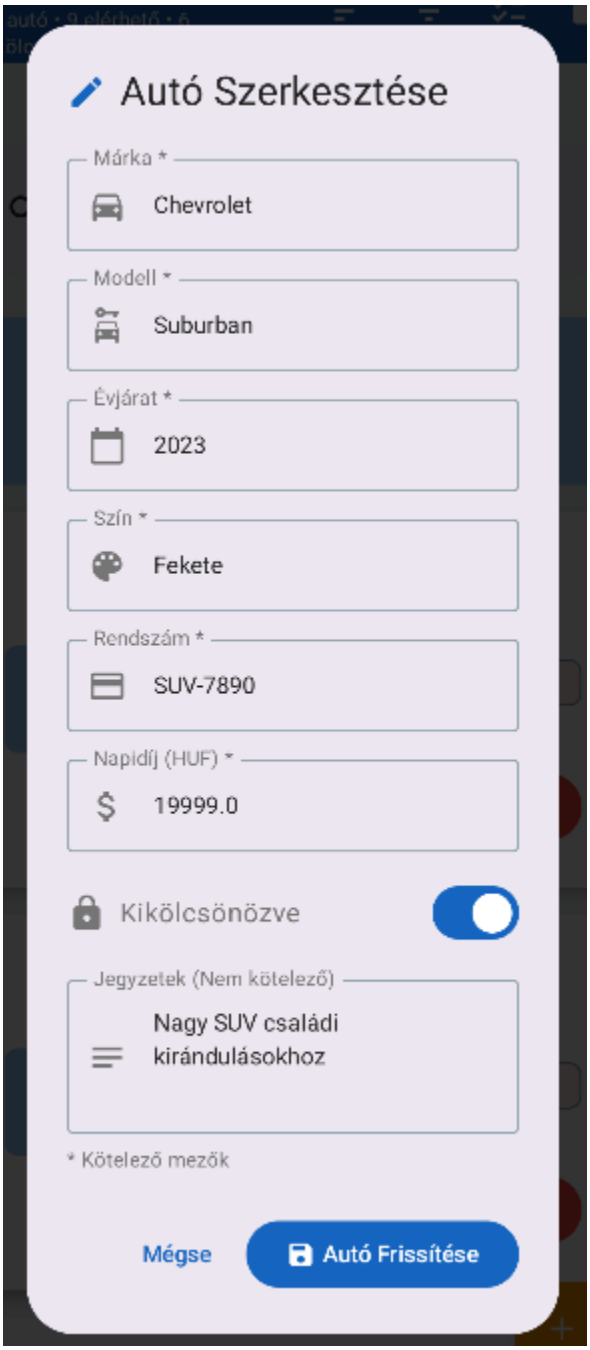
Validáció:

- Minden kötelező mező ellenőrzése
- Hibaüzenetek megjelenítése a mezők alatt
- Piros keret hibás mezőknél
- Mentés gomb csak érvényes adatokkal működik



Autó szerkesztése

Egy autó kártyáján a "Szerkesztés" gombra kattintva megnyílik a szerkesztés dialógus:



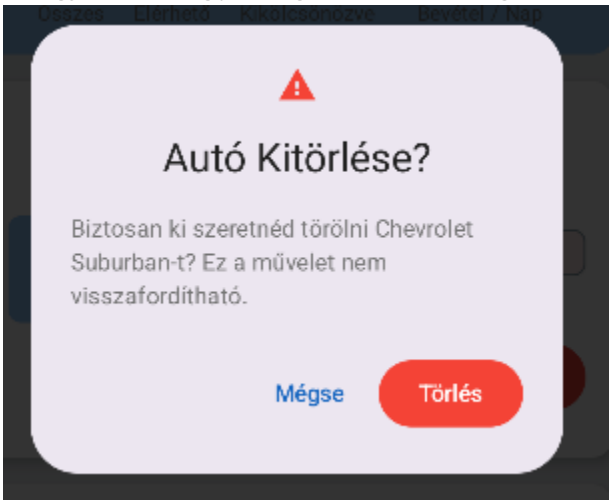
A dialógus ugyanaz, mint a hozzáadás, de előre kitöltve az autó jelenlegi adataival.

Autó törlése

Egy autó törléséhez:

1. Kattints a "Törlés" gombra az autó kártyáján

2. Megjelenik egy megerősítő dialógus

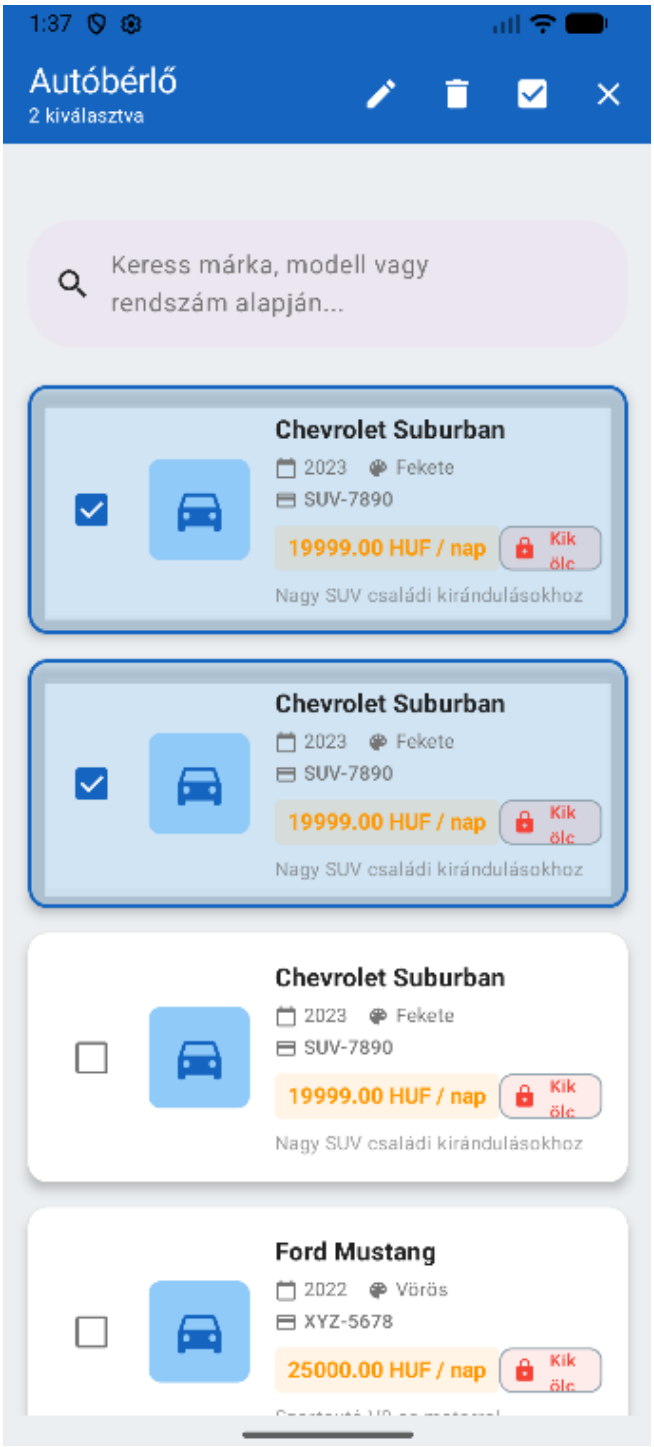


A dialógus tartalmazza:

- Figyelmeztető ikon
- "Autó Kitörlése?" cím
- Megerősítő szöveg"
- "Törlés" gomb (piros)
- "Mégse" gomb

Többszörös kiválasztás (Multi-Select Mode)

A TopAppBar-ban a checklist ikonra kattintva aktiválódik a többszörös kiválasztás mód:



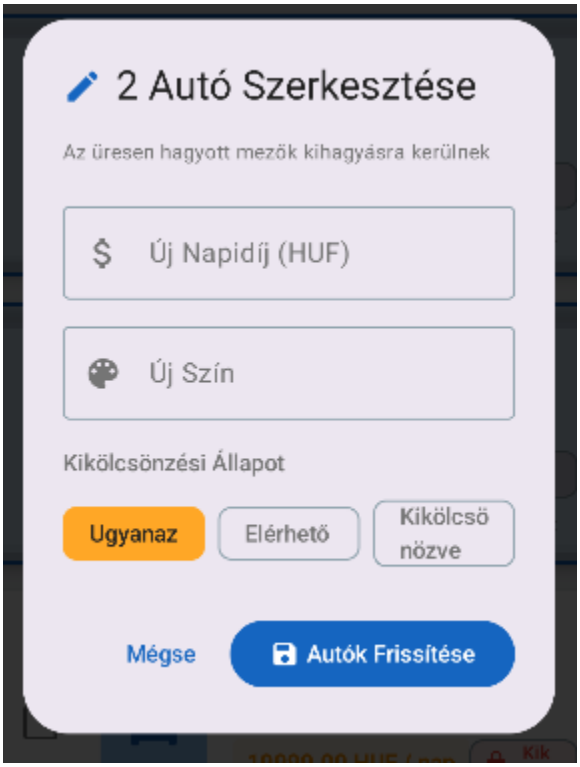
Funkciók:

- Minden autó kártyán megjelenik egy checkbox
- Kiválasztható több autó egyszerre

- TopAppBar-ban megjelenik a kiválasztott autók száma
- Új akció gombok:
- **Szerkesztés ikon**: Tömeges szerkesztés
- **Törlés ikon**: Tömeges törlés
- **Checkbox ikon**: Összes kiválasztása/kiürítése
- **X ikon**: Kilépés a multi-select módból

Tömeges szerkesztés

Több autó kiválasztása után a szerkesztés ikonra kattintva megnyílik a tömeges szerkesztés dialógus:



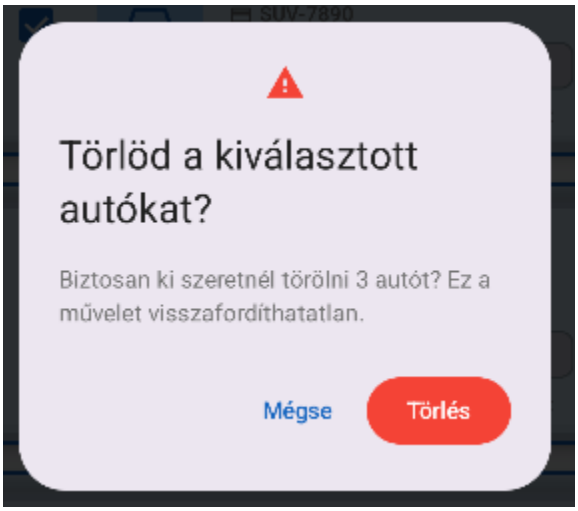
Mezők (mind opcionális, üresen hagyható):

- **Új napi díj (HUF)**: Csak akkor módosít, ha kitöltve
- **Új szín**: Csak akkor módosít, ha kitöltve
- **Bérlési státusz**:
    - "Nincs változás" (alapértelmezett)
    - "Elérhető" (minden kiválasztott autó elérhetővé válik)
    - "Kikölcsönzött" (minden kiválasztott autó kikölcsönzötté válik)

A dialógus csak a kitöltött mezőket alkalmazza a kiválasztott autókra.

Tömeges törlés

Több autó kiválasztása után a törlés ikonra kattintva megnyílik a tömeges törlés megerősítő dialógus:

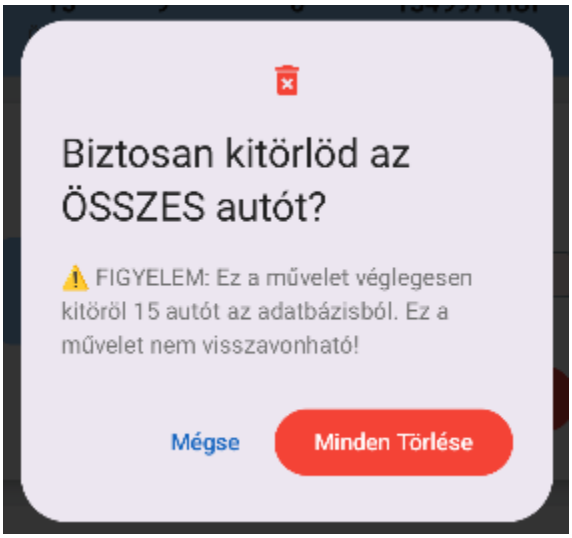


A dialógus tartalmazza:

- Figyelmeztető ikon
- "Törlöd a kiválasztott autókat?" cím
- Megerősítő szöveg
- "Törlés" gomb (piros)
- "Mégse" gomb

Összes autó törlése

A TopAppBar-ban a "Delete All" (kuka ikon) gombra kattintva megnyílik az összes autó törlésének megerősítő dialógusa:

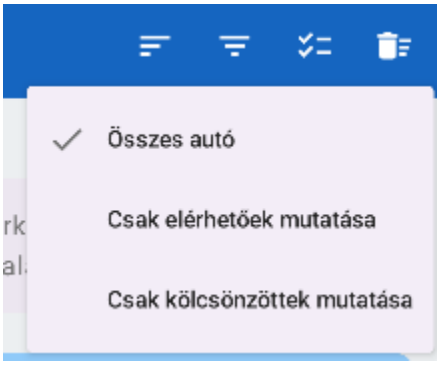


A dialógus tartalmazza:

- Figyelmeztető ikon
- "Biztosan kitörlöd az ÖSSZES autót?" cím
- Erős figyelmeztetés
- "Törlés" gomb (piros)
- "Mégse" gomb

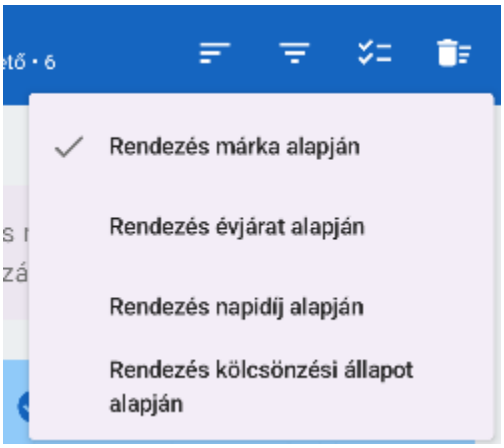
Szűrés és rendezés

Szűrés: A szűrő ikonra kattintva megnyílik egy dropdown menü:



- **Összes autó**: Minden autó megjelenítése
- **Csak elérhető**: Csak az elérhető autók
- **Csak kikölcsönzött**: Csak a kikölcsönzött autók

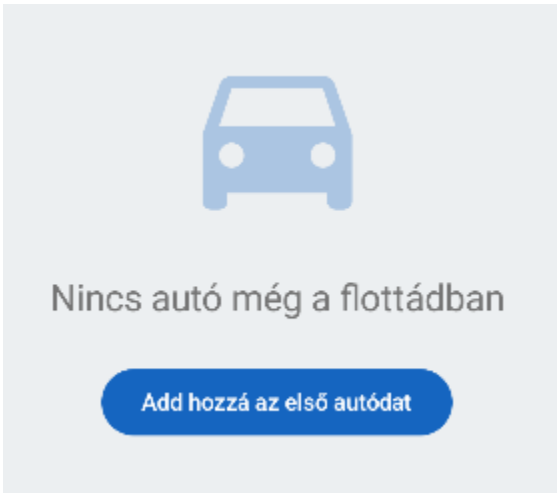
Rendezés: A rendezés ikonra kattintva megnyílik egy dropdown menü:



- **Márka szerint**: ABC sorrendben
- **Évjárat szerint**: Legújabbtól a legrégebbig
- **Napi díj szerint**: Legmagasabbtól a legalacsonyabbig
- **Bérlési státusz szerint**: Elérhető autók először

Üres állapot

Amikor nincs autó az adatbázisban, vagy a szűrő/keresés nem ad eredményt, megjelenik egy üres állapot komponens:

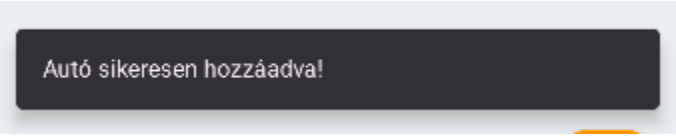


Tartalom:

- Nagy autó ikon (átlátszó kék színnel)
- Üzenet: "Nincs autó még a flottádban" (vagy megfelelő üzenet a szűrő/keresés alapján)
- "Add hozzá az első autódat" gomb (ha nincs egyetlen autó sem)

Visszajelzések

Sikeres műveletek: Minden sikeres művelet után megjelenik egy Snackbar üzenet az alján:



Példa üzenetek:

- "Autó sikeresen hozzáadva!"
- "Autó sikeresen frissítve!"
- "Autó sikeresen törölve!"
- "X autó sikeresen törölve!" (tömeges törlés)
- "Összes autó sikeresen törölve!"

6. Kód magyarázat

MVVM Architektúra

Az alkalmazás MVVM (Model-View-ViewModel) architektúrát követ:

Model (Adatmodell)
↓
Repository (Adatréteg absztrakció)
↓
ViewModel (Üzleti logika, állapotkezelés)
↓
View (UI komponensek - Compose)

CarItem Entity (Adatmodell)

```
@Entity(tableName = "car_items")
data class CarItem(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "id")
    val id: Long = 0,

    @ColumnInfo(name = "brand")
    val brand: String,

    // ... további mezők
)
```

A CarItem egy Room entitás, amely az autó adatait reprezentálja. A @Entity annotáció jelzi, hogy ez egy adatbázis tábla. A @PrimaryKey az elsődleges kulcsot jelöli, autoGenerate = true esetén automatikusan generálódik.

Validációs függvények:

```
companion object {
    fun validateBrand(brand: String): ValidationResult {
        return when {
            brand.isBlank() -> ValidationResult(false, "...")
            brand.length < 2 -> ValidationResult(false, "...")
            // ...
        }
    }
}
```

Minden mezőhöz tartozik egy validációs függvény, amely ellenőrzi az adatok érvényességét és ValidationResult objektumot ad vissza.

CarItemDao (Adatbázis hozzáférés)

```
@Dao
interface CarItemDao {
    @Query("SELECT * FROM car_items ORDER BY brand ASC, model ASC")
    fun getAllCars(): Flow<List<CarItem>>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertCar(car: CarItem): Long

    @Update
    suspend fun updateCar(car: CarItem)

    // ...
}
```

A DAO (Data Access Object) interface definiálja az adatbázis műveleteket. A @Query annotációval SQL lekérdezéseket írhatunk, a @Insert, @Update, @Delete annotációkkal pedig egyszerű műveleteket. A Flow<List<CarItem>> típus reaktív adatfolyamot biztosít, amely automatikusan frissül, amikor az adatbázis változik.

AppDatabase (Adatbázis singleton)

```
@Database(
    entities = [CarItem::class],
    version = 1,
    exportSchema = false
)
abstract class AppDatabase : RoomDatabase() {
    abstract fun carItemDao(): CarItemDao

    companion object {
        @Volatile
        private var INSTANCE: AppDatabase? = null

        fun getInstance(context: Context): AppDatabase {
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(...).build()
                INSTANCE = instance
                instance
            }
        }
    }
}
```

A AppDatabase egy singleton osztály, amely biztosítja, hogy csak egy adatbázis példány legyen az alkalmazásban. A synchronized blokk biztosítja a thread-safety-t.

CarRepository (Repository pattern)

```
class CarRepository(private val carItemDao: CarItemDao) {
    fun getAllCars(): Flow<List<CarItem>> = carItemDao.getAllCars()

    suspend fun insertCar(car: CarItem): Long =
withContext(Dispatchers.IO) {
        carItemDao.insertCar(car)
    }
}
```

A Repository absztrahálja az adatforrást, lehetővé teszi, hogy a ViewModel ne függjön közvetlenül a DAO-tól. A withContext(Dispatchers.IO) biztosítja, hogy az adatbázis műveletek az IO szálon fussanak.

CarViewModel (Állapotkezelés)

```
class CarViewModel(application: Application) :
AndroidViewModel(application) {
    private val repository: CarRepository = CarRepository(
        AppDatabase.getInstance(application).carItemDao()
    )

    private val allCarsFromDb: Flow<List<CarItem>> =
repository.getAllCars()

    val cars: StateFlow<List<CarItem>> = combine(
        allCarsFromDb,
        _currentFilter,
        _currentSort,
        _searchQuery
    ) { cars, filter, sort, query ->
        // Szűrés, rendezés, keresés logika
    }.stateIn(...)
}
```

A ViewModel kezeli az üzleti logikát és az állapotot. A StateFlow reaktív állapotkezelést biztosít, amely automatikusan frissíti a UI-t, amikor az adatok változnak. A combine függvény több Flow-ot kombinál, így a szűrés, rendezés és keresés valós időben működik.

CRUD műveletek:

```
fun addCar(brand: String, model: String, ...) {
    viewModelScope.launch {
        try {
            _isLoading.value = true
            val validationResult = CarItem.validateAll(...)
            if (!validationResult.isValid) {
                _errorMessage.value = validationResult.errorMessage
                return@launch
            }
            val car = CarItem(...)
            repository.insertCar(car)
            _successMessage.value = "Autó sikeresen hozzáadva!"
        } catch (e: Exception) {
            _errorMessage.value = "Failed to add car: ${e.message}"
        } finally {
            _isLoading.value = false
        }
    }
}
```

Minden CRUD művelet:

1. Validálja az adatokat
2. Betöltési állapotot beállít
3. Try-catch blokkban végrehajtja a műveletet
4. Sikeres/hibás üzenetet állít be
5. Végül kikapcsolja a betöltési állapotot

CarListScreen (UI komponens)

```
@Composable
fun CarListScreen(viewModel: CarViewModel = viewModel()) {
    val cars by viewModel.cars.collectAsState()
    val isLoading by viewModel.isLoading.collectAsState()
    // ...

    Scaffold(
        topBar = { /* TopAppBar */ },
        floatingActionButton = { /* FAB */ }
    ) { paddingValues ->
        if (isLoading) {
            CircularProgressIndicator()
        } else if (cars.isEmpty()) {
            EmptyState(...)
        } else {
            LazyColumn {
                items(cars) { car ->
                    CarListItem(car = car, ...)
                }
            }
        }
    }
}
```

A CarListScreen a fő UI komponens. A collectAsState() függvény a StateFlow-ot Compose State-té alakítja, így a UI automatikusan újrarajzolódik, amikor az adatok változnak. A LazyColumn hatékonyan kezeli a hosszú listákat, csak a látható elemeket rajzolja ki.

CarDialog (Hozzáadás/Szerkesztés dialógus)

```
@Composable
fun CarDialog(
    title: String,
    car: CarItem? = null,
```

```
    onDismiss: () -> Unit,
    onConfirm: (...) -> Unit
) {
    var brand by remember { mutableStateOf(car?.brand ?: "") }
    var brandError by remember { mutableStateOf<String?>(null) }

    AlertDialog(
        title = { Text(title) },
        text = {
            OutlinedTextField(
                value = brand,
                onValueChange = {
                    brand = it
                    brandError = null // Hiba törlése szerkesztéskor
                },
                isError = brandError != null,
                supportingText = brandError?.let { { Text(it) } }
            )
        },
        confirmButton = {
            Button(onClick = {
                val validation = CarItem.validateBrand(brand)
                if (!validation.isValid) {
                    brandError = validation.errorMessage
                } else {
                    onConfirm(...)
                }
            }) { Text("Mentés") }
        }
    )
}
```

A dialógus remember és mutableStateOf használatával kezeli a mezők állapotát. A validáció a "Mentés" gombra kattintáskor történik, és a hibaüzenetek a mezők alatt jelennek meg.

Többszörös kiválasztás logika

```
private val _selectedCarIds = MutableStateFlow<Set<Long>>(emptySet())
val selectedCarIds: StateFlow<Set<Long>> =
_selectedCarIds.asStateFlow()

fun toggleCarSelection(carId: Long) {
    val currentSelection = _selectedCarIds.value.toMutableSet()
    if (currentSelection.contains(carId)) {
        currentSelection.remove(carId)
    } else {
        currentSelection.add(carId)
    }
    _selectedCarIds.value = currentSelection
}
```

A kiválasztott autók ID-jait egy Set<Long> tárolja, így minden autó csak egyszer választható ki. A toggleCarSelection függvény hozzáadja vagy eltávolítja az autót a kiválasztottak közül.

7. Adatbázis struktúra

Táblaséma

```
CREATE TABLE car_items (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    brand TEXT NOT NULL,
    model TEXT NOT NULL,
    year INTEGER NOT NULL,
    color TEXT NOT NULL,
    license_plate TEXT NOT NULL,
    daily_rate REAL NOT NULL,
    is_rented INTEGER NOT NULL DEFAULT 0,
    notes TEXT NOT NULL DEFAULT ''
);
```

Mezők leírása

- **id**: Elsődleges kulcs, automatikusan generált egész szám
- **brand**: Autó márkája, szöveg, kötelező
- **model**: Autó modellje, szöveg, kötelező
- **year**: Gyártási év, egész szám, kötelező
- **color**: Szín, szöveg, kötelező
- **license_plate**: Rendszám, szöveg, kötelező, egyedi kellene legyen (jelenleg nincs constraint)
- **daily_rate**: Napi bérleti díj, valós szám, kötelező
- **is_rented**: Bérlési státusz, egész szám (0 vagy 1), alapértelmezett: 0 (false)
- **notes**: Megjegyzések, szöveg, opcionális, alapértelmezett: üres string

Lekérdezések

Összes autó lekérdezése:

```
SELECT * FROM car_items ORDER BY brand ASC, model ASC
```

Elérhető autók:

```
SELECT * FROM car_items WHERE is_rented = 0 ORDER BY brand ASC, model ASC
```

Kikölcsönzött autók:

```
SELECT * FROM car_items WHERE is_rented = 1 ORDER BY brand ASC, model ASC
```

Statisztikák:

```
SELECT COUNT(*) FROM car_items
SELECT COUNT(*) FROM car_items WHERE is_rented = 0
SELECT COUNT(*) FROM car_items WHERE is_rented = 1
SELECT SUM(daily_rate) FROM car_items WHERE is_rented = 1
```

Mintaadatok

Az alkalmazás első indításakor automatikusan betöltődik 5 minta autó