Dokumentácia

k semestrálnej práci z predmetu

Vývoj aplikácií pre mobilné zariadenia

PREHĽAD SPOTREBOVANÉHO PALIVA

vypracoval: Martin Ján Kulich
študijná skupina: 5ZYI22

dňa 9.6.2024

1. Popis a analýza

Táto časť je venovaná popisu projektu, definícii problému a porovnaniu s podobnými aplikáciami.

1.1. Motivácia

Množstvo dnešných áut je schopné zobraziť aktuálnu alebo priemernú spotrebu. Žiaľ, ani jedna s týchto možností neposkytuje presnú informáciu o spotrebovanom množstve pohonnej hmoty, či cene cesty. Aktuálna spotreba predstavuje spotrebu vozidla v danom momente, teda má krátkodobé opodstatnenie. Zatiaľ, čo priemerná spotreba je hodnota hovoriaca: pri svojich jazdách spotrebúvaš priemerne asi x l/100km. Z nej môžeme vyhodnotiť ako úsporná je naša jazda z dlhodobejšieho hľadiska. Avšak priemerná spotreba môže byť pre mnohých zavádzajúca. Napríklad, ak sme prešli 100 km a priemerná spotreba je 5 l/100km, neznamená to ešte, že sme spotrebovali 5 litrov paliva. Často spotrebujeme viac.

Pri mojich cestách by som chcel vedieť presne koľko paliva som spotreboval, a teda koľko ma daná cesta stála. Preto som sa rozhodol pre tento projekt.

1.2. Popis + operácia

Úlohou aplikácie je poskytnúť čo najpresnejší prehľad o nákladoch spojených so spotrebovaným palivom pri automobilovej preprave. Umožňuje organizáciu dvoch druhov údajov: tankovania a precestované cesty.

Pre výpočet nákladov zásadne potrebuje dve informácie. Množstvo spotrebovaného paliva s jeho cenou a k tomu príslušnú dĺžku trasy.

Množstvo spotrebovaného paliva sa dá zistiť nasledujúcim spôsobom. Nádrž vozidla sa naplní doplna. Prejde sa určitá trasa. Nádrž sa opäť naplní doplna. Množstvo natankovaného paliva počas druhého čapovania zodpovedá presne spotrebovanému palivu na danú trasu. Z tohto vyplývajú dve hlavné obmedzenia projektu:

- Pre presné pracovanie je potrebné pri každom zastavení na čerpacej stanice naplniť celú nádrž.
- Aplikácia je schopná poskytnúť presné výsledky len medzi návštevami púmp. Z nich sa vypočítavajú menej presné hodnoty pre jednotlivé cesty.

Príslušná vzdialenosť medzi tankovaniami bude vypočítaná ako súčet prislúchajúcich dĺžok ciest.

1.3. Podobné aplikácie

- Fuelio

  https://play.google.com/store/apps/details?id=com.kajda.fuelio

  Bola hlavnou inšpiráciu pri mojej tvorbe. Presne ako moja aplikácia umožňuje zadanie tankovaní a k nemu prejdených ciest. Zdá sa, že jednou z jej funkcií je aj vytvorenie trasy na základe GPS sledovania (podobne ako som aj ja chcel), avšak táto funkcia je na mojom zariadení nefunkčná alebo pre správny chod vyžaduje prídavné aplikácie z ekosystému vývojára. Pri tejto príležitosti spomeniem aj, že pri analýze v 7. týždni som v nej vedel pridať cestu bez problémov. To sa však nedá povedať o súčasnom stave, kedy sa mi akt vykonať nepodarilo. Niektoré funkcie sú schované za predplatným. Musím však pochváliť absenciu reklám.

- Zapisovač km od Driversnote a Drivvo

  https://play.google.com/store/apps/details?id=com.driversnote.driversnote

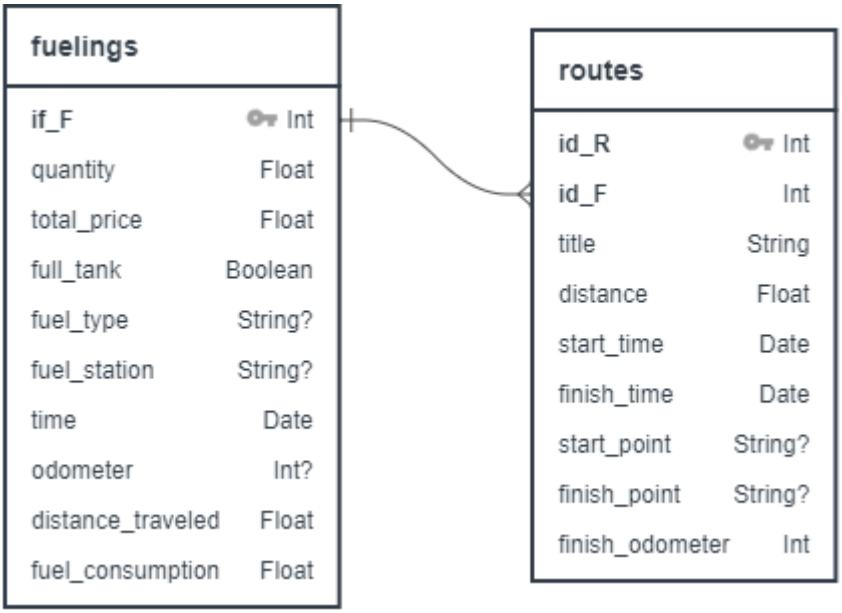
  https://play.google.com/store/apps/details?id=br.com.ctncardoso.ctncar&hl=sk

  Poskytujú služby zhodné s Fueliom. Prácu s nimi odmietam vyskúšať, pretože vyžadujú registráciu.

Uvedené aplikácie som vybral, pretože umožňujú zadávať cesty a tankovanie, s ktorých vytvárajú určitú časovú os, podobne ako moja aplikácia. Ostatné výsledky vyhľadávania v GooglePlay sú viac menej jednoduché kalkulačky spotreby.

2. Návrh riešenia problému

Ako som už spomenul aplikácia uchováva záznamy o tankovaniach a cestách. Za ich organizáciu je zodpovedná lokálna databáza, ktorej dátový diagram je uvedený nižšie. Povšimnime si, že vzťah v ňom je: n ciest k jednému tankovaniu. Teda podporuje model na jedno natankovanie sa dá precestovať viac ciest.

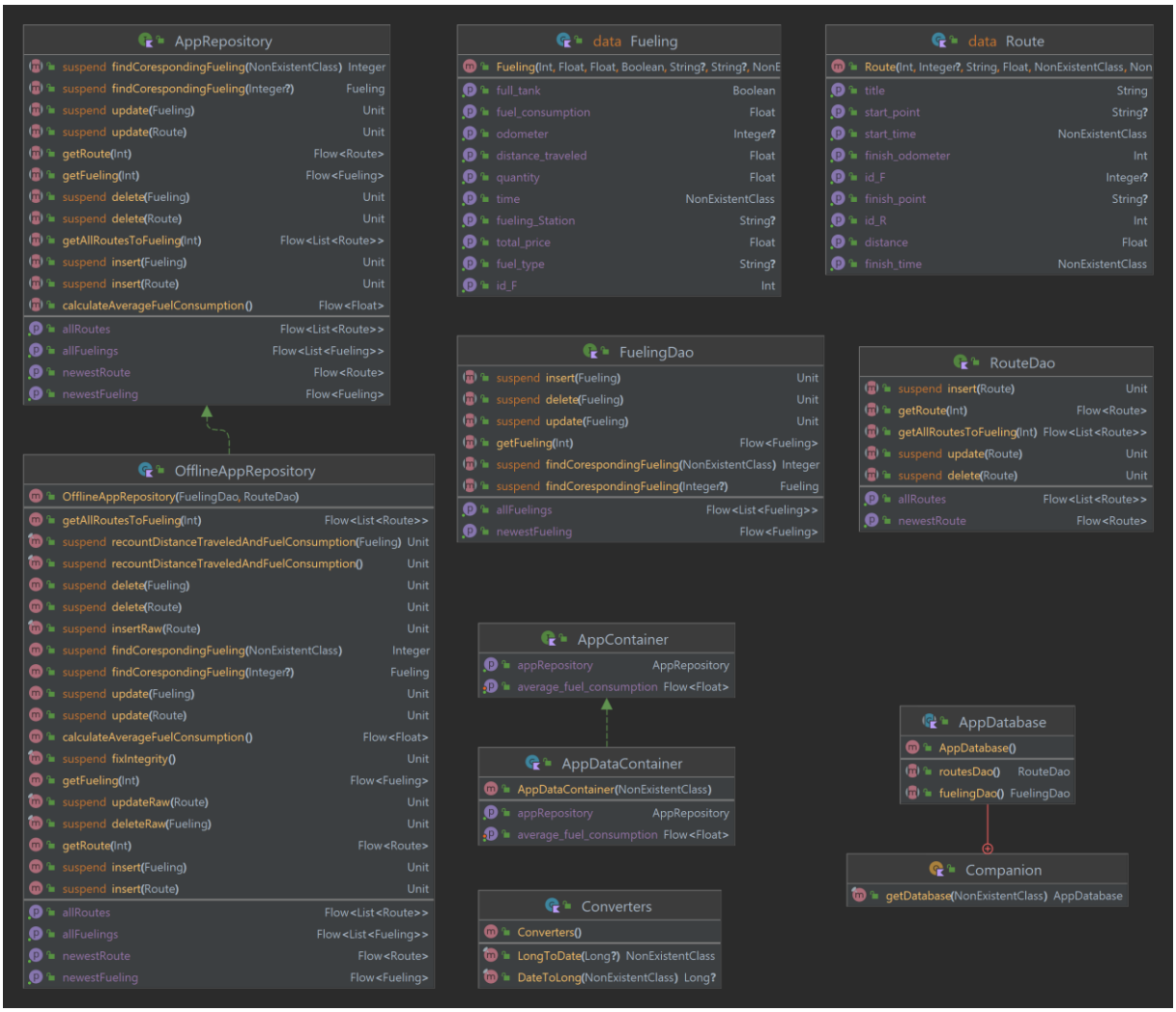


O dodržanie integrity sa stará OfflineAppRepository pracujúci s DAO vrstvou.

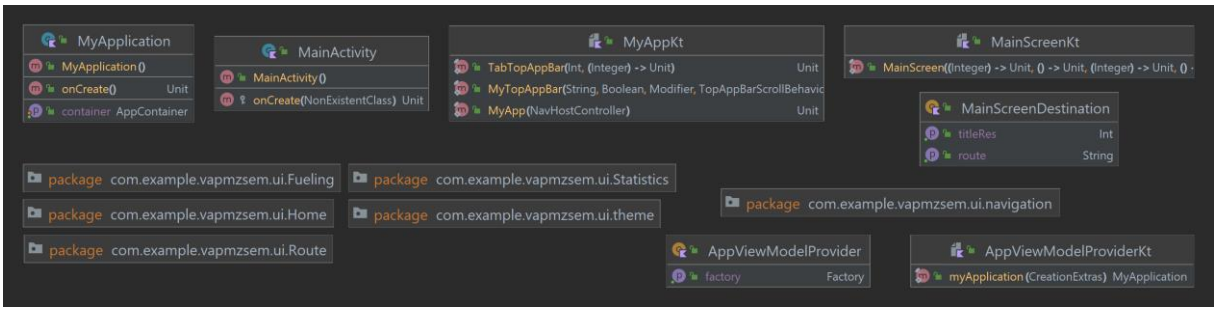
2.1. Diagram tried

Tu sú vyobrazené diagramy tried. Jeden diagram by bol moc neprehľadný, preto každý diagram zodpovedá jednému balíčku

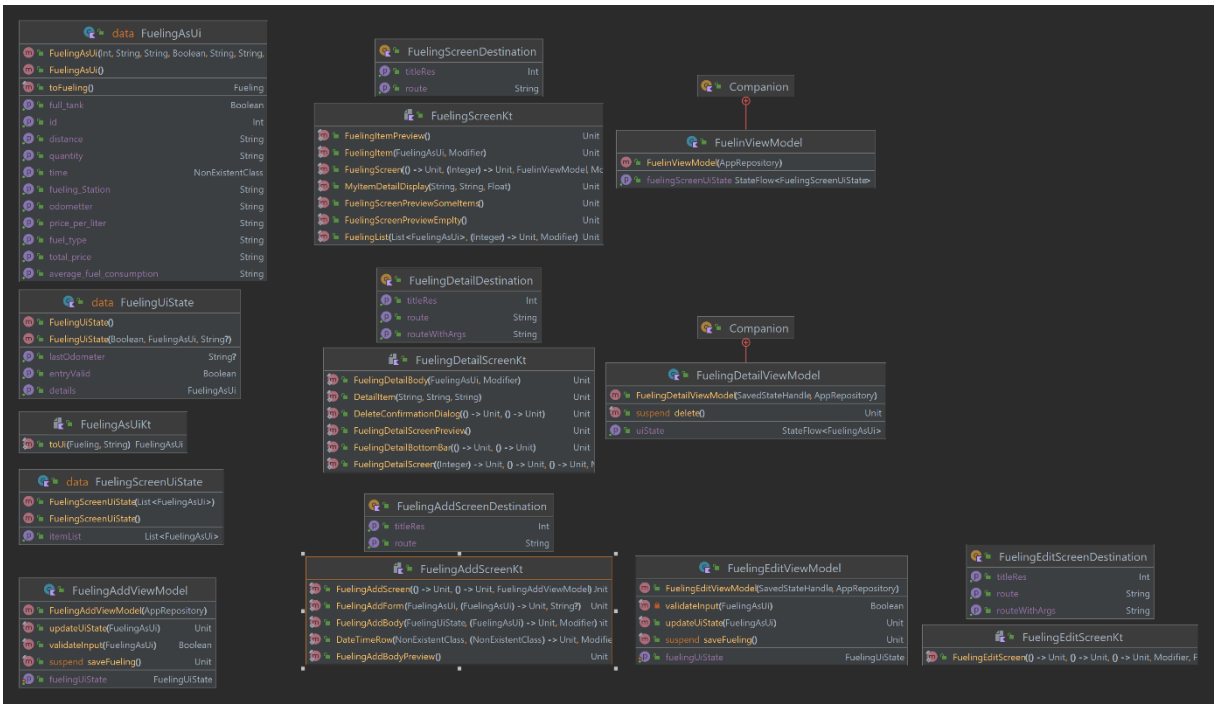
data


AppRepository
suspend findCorespondingFueling(NonExistentClass) Integer
suspend findCorespondingFueling(Integer?) Fueling
suspend update(Fueling) Unit
suspend update(Route) Unit
getRoute(Int) Flow<Route>
getFueling(Int) Flow<Fueling>
suspend delete(Fueling) Unit
suspend delete(Route) Unit
getAllRoutesToFueling(Int) Flow<List<Route>>
suspend insert(Fueling) Unit
suspend insert(Route) Unit
calculateAverageFuelConsumption() Flow<Float>
allRoutes Flow<List<Route>>
allFuelings Flow<List<Fueling>>
newestRoute Flow<Route>
newestFueling Flow<Fueling>
data Fueling
Fueling(Int, Float, Float, Boolean, String?, String?, NonE
full_tank Boolean
fuel_consumption Float
odometer Integer?
distance_traveled Float
quantity Float
time NonExistentClass
fueling_Station String?
total_price Float
fuel_type String?
id_F Int
data Route
Route(Int, Integer?, String, Float, NonExistentClass, Non
title String
start_point String?
start_time NonExistentClass
finish_odometer Int
id_F Integer?
finish_point String?
id_R Int
distance Float
finish_time NonExistentClass
FuelingDao
suspend insert(Fueling) Unit
suspend delete(Fueling) Unit
suspend update(Fueling) Unit
getFueling(Int) Flow<Fueling>
suspend findCorespondingFueling(NonExistentClass) Integer
suspend findCorespondingFueling(Integer?) Fueling
allFuelings Flow<List<Fueling>>
newestFueling Flow<Fueling>
RouteDao
suspend insert(Route) Unit
getRoute(Int) Flow<Route>
getAllRoutesToFueling(Int) Flow<List<Route>>
suspend update(Route) Unit
suspend delete(Route) Unit
allRoutes Flow<List<Route>>
newestRoute Flow<Route>
OfflineAppRepository
OfflineAppRepository(FuelingDao, RouteDao)
getAllRoutesToFueling(Int) Flow<List<Route>>
suspend recountDistanceTraveledAndFuelConsumption(Fueling) Unit
suspend recountDistanceTraveledAndFuelConsumption() Unit
suspend delete(Fueling) Unit
suspend delete(Route) Unit
suspend insertRaw(Route) Unit
suspend findCorespondingFueling(NonExistentClass) Integer
suspend findCorespondingFueling(Integer?) Fueling
suspend update(Fueling) Unit
suspend update(Route) Unit
calculateAverageFuelConsumption() Flow<Float>
suspend fixIntegrity() Unit
getFueling(Int) Flow<Fueling>
suspend updateRaw(Route) Unit
suspend deleteRaw(Fueling) Unit
getRoute(Int) Flow<Route>
suspend insert(Fueling) Unit
suspend insert(Route) Unit
allRoutes Flow<List<Route>>
allFuelings Flow<List<Fueling>>
newestRoute Flow<Route>
newestFueling Flow<Fueling>
AppContainer
appRepository AppRepository
average_fuel_consumption Flow<Float>
AppDataContainer
AppDataContainer(NonExistentClass)
appRepository AppRepository
average_fuel_consumption Flow<Float>
Converters
Converters()
LongToDate(Long?) NonExistentClass
DateToLong(NonExistentClass) Long?
AppDatabase
AppDatabase()
routesDao() RouteDao
fuelingDao() FuelingDao
Companion
getDatabase(NonExistentClass) AppDatabase

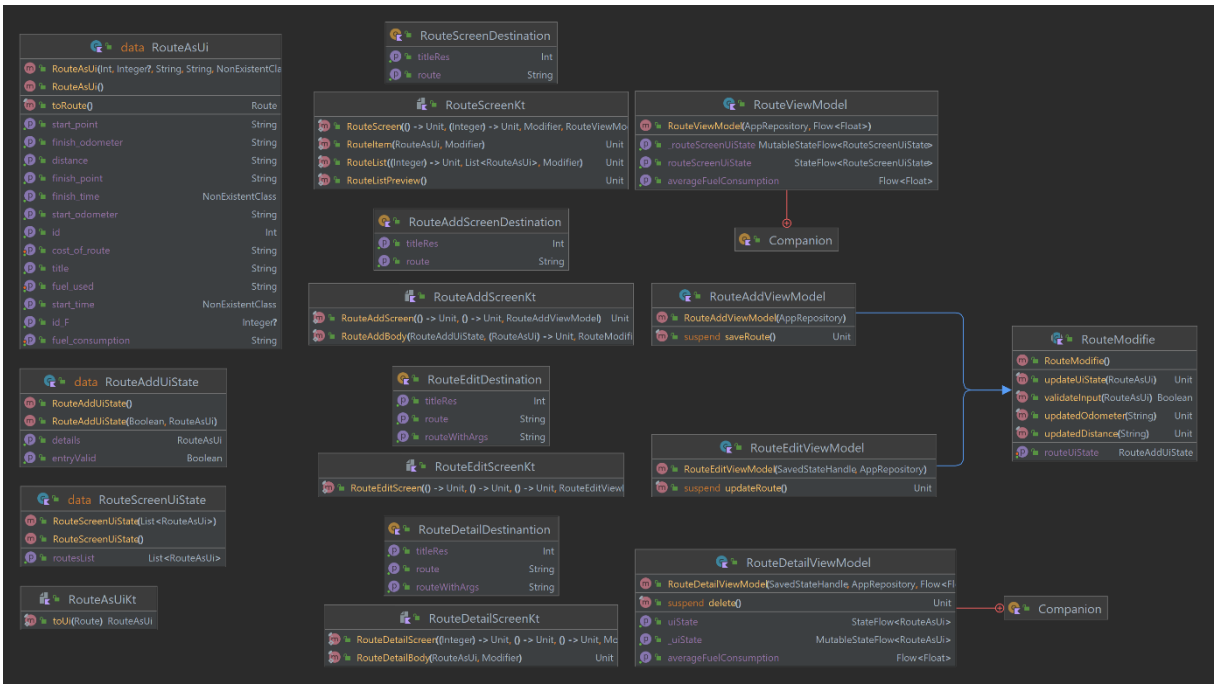
ui


MyApplication
MyApplication()
onCreate() Unit
container AppContainer
MainActivity
MainActivity()
onCreate(NonExistentClass) Unit
MyAppKt
TabTopAppBar(Int, (Integer) -> Unit) Unit
MyTopAppBar(String, Boolean, Modifier, TopAppBarScrollBehavic
MyApp(NavHostController) Unit
MainScreenKt
MainScreen((Integer) -> Unit, () -> Unit, (Integer) -> Unit, () -
MainScreenDestination
titleRes Int
route String
package com.example.vapmzsem.ui.Fueling
package com.example.vapmzsem.ui.Statistics
package com.example.vapmzsem.ui.navigation
package com.example.vapmzsem.ui.Home
package com.example.vapmzsem.ui.theme
package com.example.vapmzsem.ui.Route
AppViewModelProvider
factory Factory
AppViewModelProviderKt
myApplication(CreationExtras) MyApplication

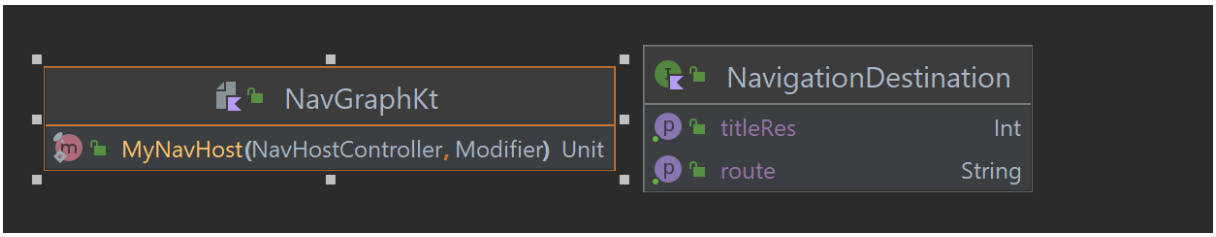
ui/Fueling



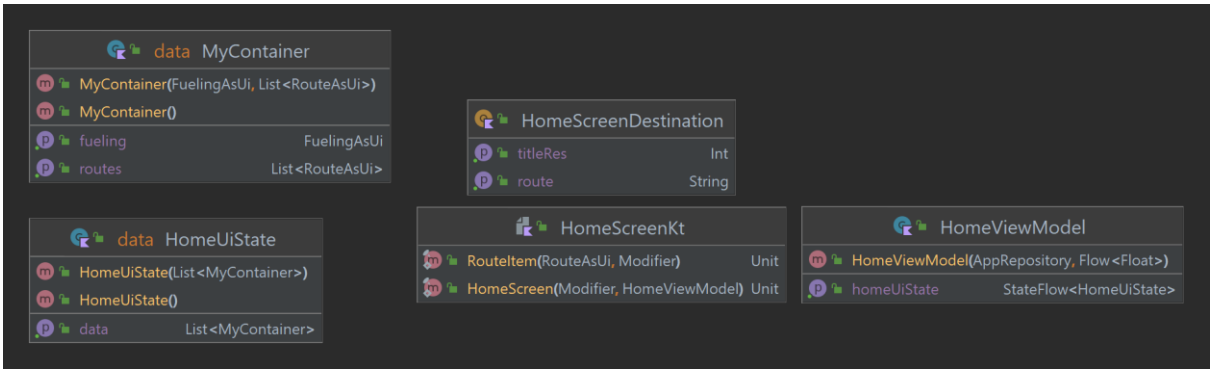
ui/Route



ui/navigation



ui/Home



3. Popis implementácie

LifeCycles – využívam na vytvorenie a správu životného cyklu view modelov. Primárne však na dependenci injection. Tento prístup zabezpečuje tiež efektívne riadenie zdrojov.

Príklad:

```
object AppViewModelProvider {
    val Factory = viewModelFactory {
        . . .
         // FuelongDetail
         initializer {
             FuelingDetailViewModel(
                 this.createSavedStateHandle(),
                 myApplication().container.appRepository
             )
         }
         . . .
}
```

```
@Composable
fun FuelingDetailScreen(
    . . .,
    viewModel: FuelingDetailViewModel  = viewModel(factory =
AppViewModelProvider.Factory)
){ . . . }
```

Navigation – využitý na navigáciu medzi obrazovkami. Každá ním navigovateľná obrazovka má svoj príslušný objekt implementujúci rozhranie NavigationDestination, podľa ktorého sa k nej naviguje.

Room – využitý na prácu s lokálnou databázou. Implementovaný podľa codelabu z 8. týždňa semestra. Umožňuje reaktívne aktualizácie UI pri zmene v databáze.

ViewModel – využitý na perzistentné držanie dát pre UI a enkapsuláciu biznis logiky. V podstatne pomáha každej obrazovke v aplikácií.