ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia
a informatiky

Semestrálna práca z predmetu vývoj aplikácií pre mobilné zariadenia – HPG system

Fakulta:	Fakulta riadenia a informatiky
Program:	Informatika
Vypracoval:	Félix Papiernik
Navštevované cvičenie:	Streda, blok 7 – 9
Vyučujúci:	doc. Ing. Patrik Hrkút, PhD., Ing. Michal Ďuračík PhD.
Školský rok:	2023/2024

Popis a analýza riešeného problému

Definovanie problému

V oblasti fitness vidím veľké množstvo dezinformácií – influencerov, čo predávajú produkty, ktoré nielenže zbytočne vyprázdnia peňaženky, ale v niektorých prípadoch aj ľudom vážne poškodia zdravie. Hlavne vidím, že športová gramotnosť ľudí sa zhoršuje – lenivieme, tučnieme, menej kráčame, vďaka tomu nám mužom klesá testosteron, máme potom menej energie do života, chradneme a v starobe sme vďaka zanedbávaniu nášho tela náchylní na choroby.

Poznám jedného skvelého trénera, ktorý mení životy ľudí k lepšiemu nielen po fyzickej, ale aj psychickej stránke, len má už priveľa klientov a málo času. Proces **práce s klientami** prebieha v mnohých aplikáciach **(excel, gmail, facebook, Inbody...**) a jeho systém obsahuje veľa monotónnych činností, ktoré ho **ukracujú o čas**.

Špecifikácia zadania

Zadanie teda predstavuje **mobilnú aplikáciu pre trénerov vo fitness centre** Universal Training Centre. Tréneri si budú môcť pridávať alebo upravovať klientov a ich merania – konkrétne ich diagnostické merania telesnej kompozície. Cielové zariadenie je android mobil, ktorý tréner bude používať vo fitness centre na zaznamenávanie výsledkov meraní zverencov.

Podobné aplikácie

GIMIFY coach

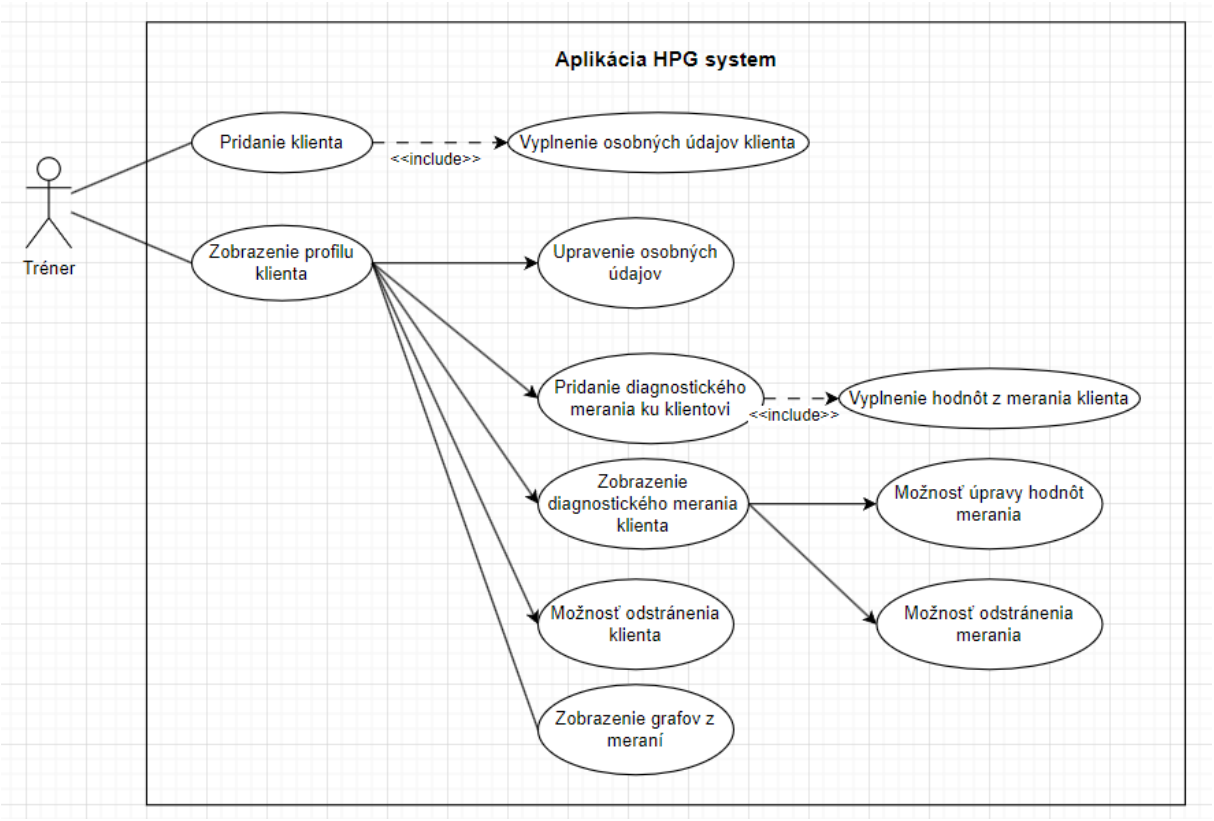
Tento produkt predstavuje lokálnu a neudržiavanú konkurenciu, ktorá je v podstate neznáma (235 sledovateľov na instagrame, 117 na facebooku). Poskytuje iba základný systém pre trénera, ale nemá dostatok funkcionalít – chýba tvorba jedálničkov a sledovanie merateľných výsledkov zverencov a ani nemá recenzie ku trénerom. V mojej aplikácii to zatiaľ neni, ale v rámci bakalárskej práce to bude.

TRUECOACH

Tento produkt, ktorý je zároveň najväčšou konkurenciou, poskytuje takmer všetko to, čo chcem vytvoriť v zadaní mojej bakalárskej práce, ale chýba tomu jedna podstatná vec – záruka, že keď si bežný človek nájde trénera, môže to byť neprofesionál a poškodiť zdravie zverenca. A ešte k tomu nemajú systém na tvorbu jedálničkov, časovač intervalov ku tréningom, diagnostiku telesnej kompozície cez inbody.

Návrh riešenia problému

Krátka analýza - diagram prípadov použitia



Návrh aplikácie

Tried je v semestrálnej práci priveľa, preto som sa rozhodol, že tento bod spracujem ako hierarchický zoznam významných balíčkov a súborov v **“\app\src\main”:**

- **java**
  - com.example.inventory
    - **data**
      - **client –** potrebné súbory pre prácu s fyzickým klientom
        - **Client.kt –** reprezentuje klienta
        - ClientDao.kt
        - ClientsRepository
        - OfflineClientsRepository
      - **measurement -** potrebné súbory pre prácu s meraním
        - **Measurement.kt –** predstavuje diagnostické meranie daného klienta
        - Measurement.kt
        - MeasurementDao.kt
        - MeasurementsRepository.kt
        - OfflineMeasurementsRepository.kt

        - AppContainer.kt
        - **HpgDatabase.kt – singleton trieda,** obsahuje databázu typu **RoomDatabase,** ktorá obsahuje entity typu **Client, Measurement**
    - **ui**
        - **client**
            - **details, edit, entry** – obrazovky pre prácu s klienmi
        - **components –** package, ktorý obsahuje **znovupoužiteľné** časti kódu, ktoré sa používajú na **viacerých miestach** v aplikácií. **Zabezpečujú konzistentný dizajn a funkcionalitu.**
        - **home –** domovská obrazovka, obsahuje zoznam klientov s možnosťou pridania nových klientov
        - **measurments**
            - **details, edit, entry** – obrazovky pre prácu s diagnostickými meraniami klientov
        - **navigation**
            - **HpgNavGraph.kt –** obsahuje všetky routes, čiže ku textovému reťazcu priradené obrazovky, ktoré sa majú v danej route vykresliť
            - NavigationDestination.kt
        - theme
        - **AppViewModelProvider.kt –** návrhový vzor **factory –** vytvára inštancie ViewModel-ov pre jednotlivé obrazovky
    - **MainActivity.kt –** inicializuje HpgApp
    - **HpgApp.kt –** top level composable, definuje komponent HpgTopAppBar, ktorý sa preiebžne používa v aplikácii
    - **HpgApplication.kt –** vytvorí inštanciu AppDataContainer typu AppContainer, ktorý v aplikácii používam ako kontajner na dependency injection
- **res**
    - drawable - obrázky
    - mipmap-anydpi-v26 - ikony
    - **values**
        - dimens.xml – nastavenie rozmerov, napríklad pre padding
        - string.xml – obsahuje texty, ktoré sa zobrazujú v UI v aplikácii
        - themes.xml
    - AndroidManifest.xml

Popis implementácie

Implementácia aplikácie začala tak, že som použil cvičenie z codelabu, ktorý sme vypracovávali na cvičení.

Následne som si poriadne popozeral ten kód, aby som zistil ako funguje a potom som začal **s pridávaním vlastnej funcionality.**

Semestrálna práca obsahuje/implementuje (ak som na niečo nezabudol):

- **7 obrazoviek, ku každej ViewModel**
- **Návrhové vzory:**
    - Singleton
    - Factory
    - Repository
    - Dependency injection
- **Navigation**
- **Room** databázu
- **Knižnicu** Ycharts na prácu s grafmi, ktoré vykresľujú prehľad diagnostických meraní

Aplikácia je naprogramovaná v jazyku kotlin s použitím jetpack compose. Začal by som teda triedou **HpgApp.kt** a nasledne prešiel ku ostatným dôležitým častiam kódu.

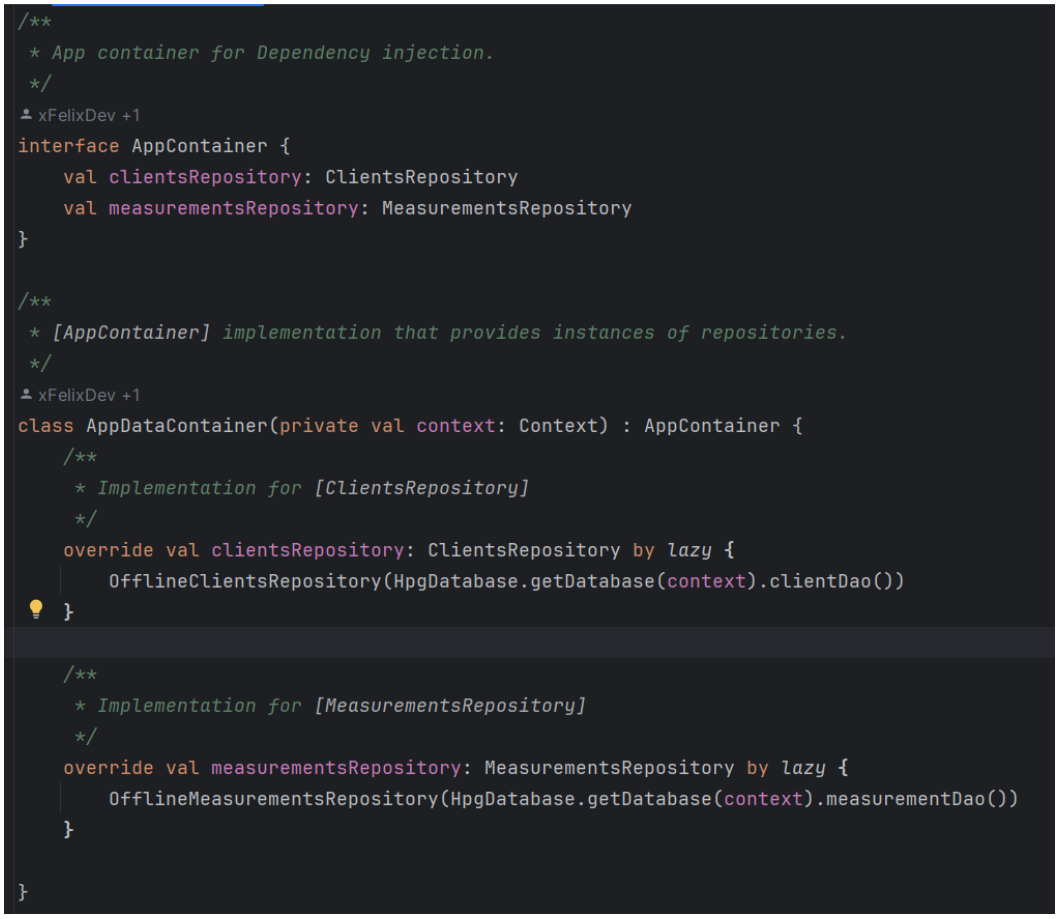
- **HpgApp.kt**
    - definuje znovupoužiteľný komponent – HpgTopAppBar, ktorý sa používa na každej obrazovke ako vrchná lišta, kde sa zobrazuje nadpis – názov danej obrazovky, na ktorej sa nachádzame.
    - vytvára inštanciu HpgNavHost typu **NavHost** zo súboru HpgNavGraph.kt
- **HpgNavGraph.kt**
    - definuje HpgNavHost composable
    - v nej sú zadefinované všetky routes, čiže cesty prislúchajúce danej obrazovke, tejto aplikácie
    - na začiatku vykreslí domovskú obrazovku

```
@Composable
fun HpgNavHost(
    navController: NavHostController,
    modifier: Modifier = Modifier,
) {
    NavHost(
        navController = navController,
        startDestination = HomeDestination.route,
        modifier = modifier
    ) { this: NavGraphBuilder
        composable(route = HomeDestination.route) { this: AnimatedContentScope  it: NavBackStackEntry
            HomeScreen(
                navigateToClientEntry = { navController.navigate(ClientEntryDestination.rou
                navigateToClientUpdate = { it: Int
                    navController.navigate( route: "${ClientDetailsDestination.route}/${it}")
                },
            )
        }
        composable(route = ClientEntryDestination.route) { this: AnimatedContentScope  it: NavBackStac
            ClientEntryScreen(
                navigateBack = { navController.popBackStack() },
                onNavigateUp = { navController.navigateUp() }
            )
        }
        composable(
            route = ClientDetailsDestination.routeWithArgs,
            arguments = listOf(navArgument(ClientDetailsDestination.clientIdArg) { this: NavArg
                type = NavType.IntType
            })
```

- zoznam jednotlivých composale pre prislúchajúce routy pokračuje v kóde

- **AppViewModelProvider.kt**
  - obsahuje atribút implementujúci **Factory** návrhový vzor, ten je zodpovedný za vytváranie inštancií rôznych ViewModelov, takto jednotlivé obrazovky nepotrebujú poznať všetky parametre, ktoré ViewModel potrebuje – ani to predsa nie je zodpovednosť tej obrazovky.

```
/**
 * Provides Factory to create instance of ViewModel for the entire HPG app
 */
xFelixDev +1
object AppViewModelProvider {
    val Factory = viewModelFactory { this: InitializerViewModelFactoryBuilder | creationExtras: Cre
        // Initializer for HomeViewModel
        initializer { this: CreationExtras
            HomeViewModel(inventoryApplication().container.clientsRepository)
        }
        // Initializer for ClientEntryViewModel
        initializer { this: CreationExtras
            ClientEntryViewModel(inventoryApplication().container.clientsRepository)
        }
        // Initializer for ClientDetailsViewModel
        initializer { this: CreationExtras
            ClientDetailsViewModel(
                this.createSavedStateHandle(),
                inventoryApplication().container.clientsRepository,
                inventoryApplication().container.measurementsRepository,
            )
        }
        // Initializer for ClientEditViewModel
        initializer { this: CreationExtras
            ClientEditViewModel(
                this.createSavedStateHandle(),
                inventoryApplication().container.clientsRepository
            )
```

- **AppContainer.kt**
  - implementuje **dependency injection** tak, že pre aplikáciu poskytuje potrebné objekty – clientsRepository a measurementsRepository, namiesto toho aby ich sama priamo vytvorila.
  - aplikácia si je týmto schopná jednoducho vypýtať potrebné údaje
  - 

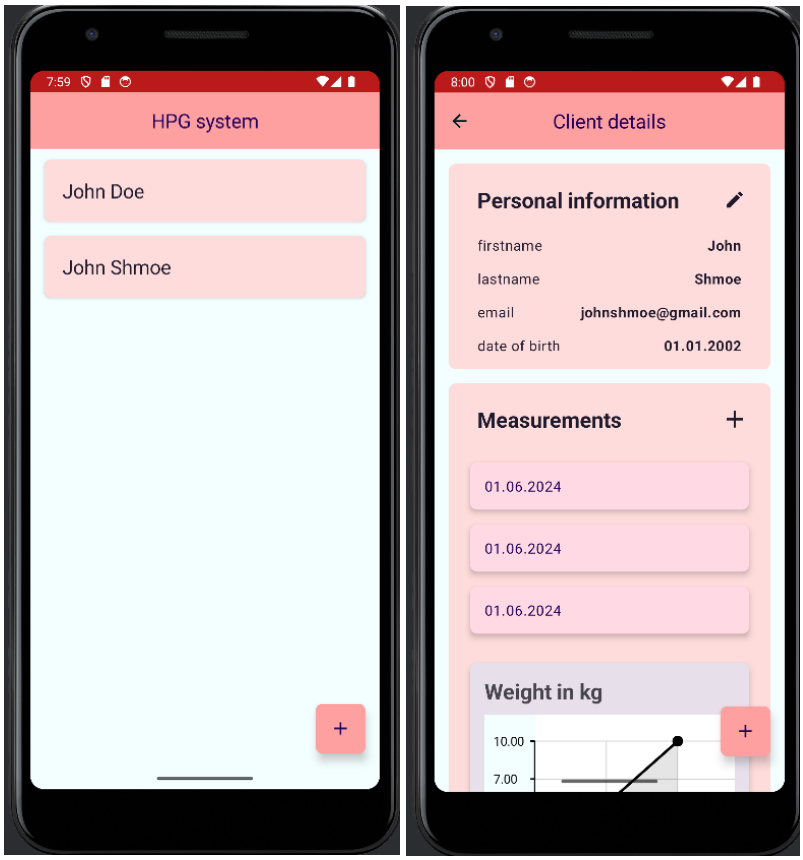
```kotlin
/**
 * App container for Dependency injection.
 */
xFelixDev +1
interface AppContainer {
    val clientsRepository: ClientsRepository
    val measurementsRepository: MeasurementsRepository
}

/**
 * [AppContainer] implementation that provides instances of repositories.
 */
xFelixDev +1
class AppDataContainer(private val context: Context) : AppContainer {
    /**
     * Implementation for [ClientsRepository]
     */
    override val clientsRepository: ClientsRepository by lazy {
        OfflineClientsRepository(HpgDatabase.getDatabase(context).clientDao())
    }

    /**
     * Implementation for [MeasurementsRepository]
     */
    override val measurementsRepository: MeasurementsRepository by lazy {
        OfflineMeasurementsRepository(HpgDatabase.getDatabase(context).measurementDao())
    }

}
```

- **HpgDatabase.kt**
  - **Room databáza,** ktorá obsahuje **entity – tabuľky:** client, measurement
  - poskytuje aj **DAO** objekty pre jednoduchý prístup k dátam databázy
- **OfflineClientsRepository , OfflineMeasurementsRepository**
  - poskytujú operácie pre prácu s databázou obsahujúci **measurements** a **clients**
- **Obrazovky a ich ViewModely**
  - **HomeScreen**, **HomeViewModel** – domovská obrazovka zobrazujúca zoznam klientov a poskytuje akciu, ktorá prejde na novú obrazovku, pomocou ktorej sa môže pridať nový klient. HomeViewModel sa stará o údajovú časť – o zoznam klientov.
  - **ClientDetailsScreen, ClientDetailsViewModel** – obrazovka zobrazujúca osobné údaje klienta, zoznam meraní a grafy zobrazujúce pokrok kliento v hodnotách meraní. ClientDetailsViewModel sa stará o údaje o klientovi a zozname meraní.



- **ClientEntryScreen –** zobrazuje obrazovku, ktorá slúži na vytvorenie nového klienta. Používateľ vyplní údaje a až po **správnom** zadaní všetkých údajov vytvorí klienta. **ClientEntryViewModel** sa stará o zachovanie dát a o validáciu údajov – validáciu emailu a dátumu narodenia.

```
fun validateClientDetailsInput(uiState: ClientDetails): Boolean {
    return with(uiState) { this: ClientDetails
        firstName.isNotBlank() && lastName.isNotBlank()
                && email.isNotBlank() && dateOfBirth.isNotBlank() &&
                email.contains( other: "@") && isValidDate(dateOfBirth)
    }
}
 xFelixDev
fun validEmail(email: String): Boolean {
    return email.matches(Regex( pattern: "[a-zA-Z0-9._-]+@[a-z]+\\.+[a-z]+"))
}
```

- **ClientEditScreen –** poskytuje možnosť úpravy osobných údajov už existujúceho klienta. **ClientEditViewModel** sa stará o dátovú časť tejto obrazovky.
- **MeasurementEntryScreen –** poskytuje používateľovi možnosť pridať diagnostické meranie ku vybranému klientovi. **MeasurementEntryViewModel**

spracováva dátovú časť, kontroluje či je všetko čo má zadané a validuje údaje.

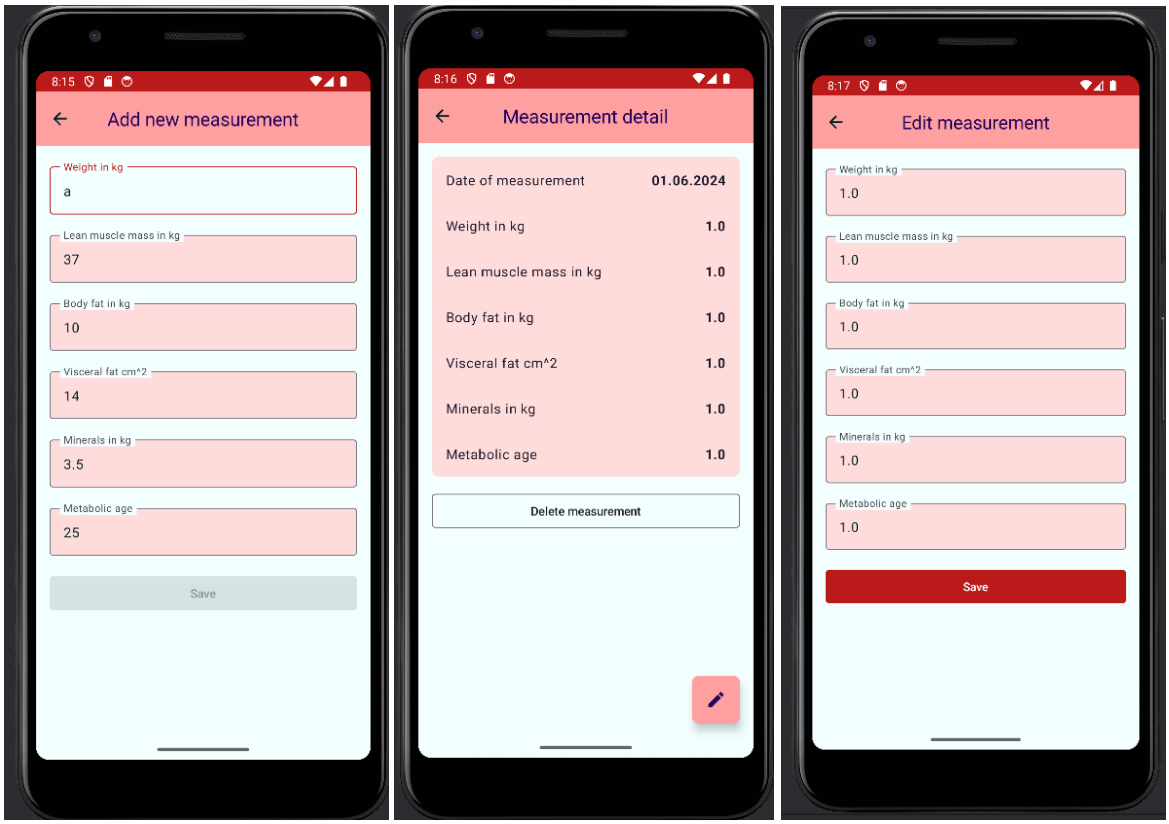
```
    xFelixDev
    suspend fun saveMeasurement() {
        if (validateMeasurementInput(measurementUiState.measurementDetails)) {
            measurementsRepository.insertMeasurement(measurementUiState.measurementDetails.toMe
        }
    }
}

xFelixDev
fun validateMeasurementInput(uiState: MeasurementDetails): Boolean {
    return with(uiState) { this: MeasurementDetails
        bodyWeightKg.isNotEmpty() && bodyWeightKg.toDoubleOrNull() != null
                && leanMuscleMassKg.isNotEmpty() && leanMuscleMassKg.toDoubleOrNull() != null
                && bodyFatKg.isNotEmpty() && bodyFatKg.toDoubleOrNull() != null
                && visceralFat.isNotEmpty() && visceralFat.toDoubleOrNull() != null
                && mineralsKg.isNotEmpty() && mineralsKg.toDoubleOrNull() != null
                && metabolicAge.isNotEmpty() && metabolicAge.toDoubleOrNull() != null
    }
}
```

- **MeasurementEditScreen** poskytuje možnosť úpravy už existujúceho merania, zaujímavosťou je, že MeasurementEditViewModel znovupoužíva validáciu údajov, ktorá je zadefinovaná v MeasurementEntryScreen.

```
    /**
     * Update the measurement in the [MeasurementsRepository]'s data source
     */
    xFelixDev
    suspend fun updateMeasurement() {
        if (validateMeasurementInput(measurementUiState.measurementDetails)) {
            measurementsRepository.updateMeasurement(measurementUiState.measurementDetails.toMeasureme
        }
    }
```

- **MeasurementDetailsScreen** zobrazuje údaje o vybranom meraní, poskytuje možnosť úpravy alebo odstránenia



Pri vývoji aplikácie som pracoval s githubom, pričom dokopy mám cez 20 commitov a na projekte som pracoval viac ako 6 týždňov.