ŽILINSKÁ UNIVERZITA V ŽILINE

Fakulta riadenia
a informatiky

Fakulta riadenia a informatiky

Semestrálna práca

Vývoj aplikácií pre mobilné zariadenia

CalorieTracker

Samuel Lesko

Št. skupina: 5ZYR21

Šk rok: 2023/2024

1. Popis a analýza riešeného problému

a. Špecifikácia zadania

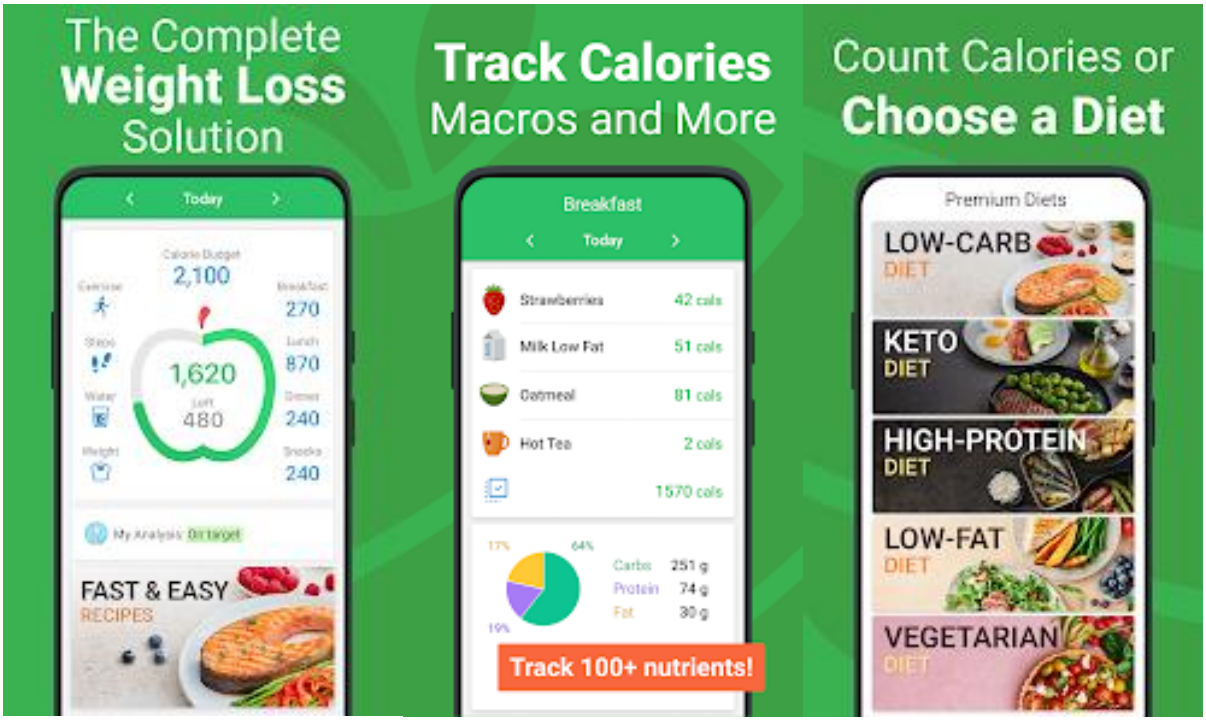
Aplikácia CalorieTracker by mala byť android aplikácia, ktorá umožňuje užívateľovi sledovať svoj denný príjem kalórií.
Okrem toho aplikácia zobrazuje aj nutričné informácie, ako sú bielkoviny, sacharidy a tak ďalej.
Aplikácia by mala umožňovať teda monitorovanie rôznych štatistík.
Aplikácia by mala byť prispôsobená na konkrétneho používateľa.
Teda podľa jeho veku, váhy, výšky a pohlavia by mu mala prepočítať aké množstvo kalórií by mal prijať.
Aplikácia by mala používateľovi dovoliť vyhľadávať v zozname jedál a následne kliknutím na dané jedlo zobraziť údaje o kalóriach a nutričných hodnotách tohto jedla.
Toto zobrazenie slúži na to aby si mohol používateľ toto jedlo potom ako ho zjedol pridať do svojho denného jedálníčka, s tým že zadá aké množstvo tohto jedla zjedol.
Údaje o jedle sa potom prepočítajú a uložia.
Všetky jedlá, ktoré používateľ zjedol sa aj zobrazujú na ďalšej obrazovke, kde si môže používateľ pozrieť, aké hodnoty mu dané jedlo pridalo.
Celkové hodnoty sú potom pre používateľa zobrazené na prvej obrazovke.
Aplikácia je vytvorená tak aby sa používateľovi vždy na nový deň vymazali všetky zjedené jedlá a používateľ si mohol znova v nový deň sledovať svoj príjem kalórií a iných nutričných hodnôt.

b. Podobné aplikácie

Tu si teraz popíšeme podobné aplikácie, ako tá ktorú som vytvoril, ktoré môžete nájsť na google store. Pod nimi je potom ukážka mojej aplikácie a popis v čom sa líši od popísaných existujúcich aplikácií.

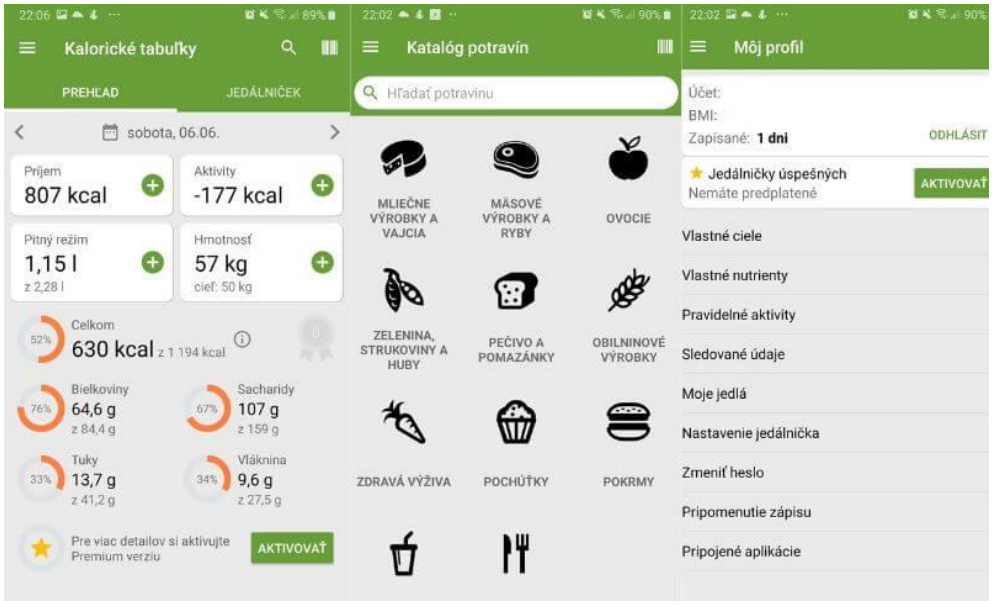
i. Calorie Counter – MyNetDiary

CalorieCounter - MyNetDiary je aplikácia na sledovanie stravy a na výpočet kalórií, ktorá pomáha ľuďom dosiahnuť svoje ciele týkajúce sa stravovania a telesnej hmotnosti.
Umožňuje užívateľom sledovať svoj príjem kalórií.
Užívatelia môžu sledovať svoj príjem kalórií a nutričných hodnôt jedál a nápojov. Aplikácia teda obsahuje aj databázu potravín.
Aplikácia umožňuje užívateľom sledovať ich fyzickú aktivitu vrátane cvičenia, krokov, prejdeného vzdialenosti a iných cvičebných aktivít. To pomáha užívateľom udržať si celkovú rovnováhu medzi príjmom kalórií a ich spaľovaním.
Užívatelia taktiež môžu sledovať svoju váhu v čase a zaznamenávať pokrok vo svojom cieľovom hmotnostnom rozsahu.
Aplikácia poskytuje užívateľom prístup k rôznym štatistikám a grafom, ktoré zobrazujú ich pokrok v dosahovaní cieľov týkajúcich sa stravovania a cvičenia.
Je to komplexná aplikácia, ktorá poskytuje užívateľom nástroje na sledovanie ich stravovacích a cvičebných návykov a pomáha im dosiahnuť ich zdravotné ciele.



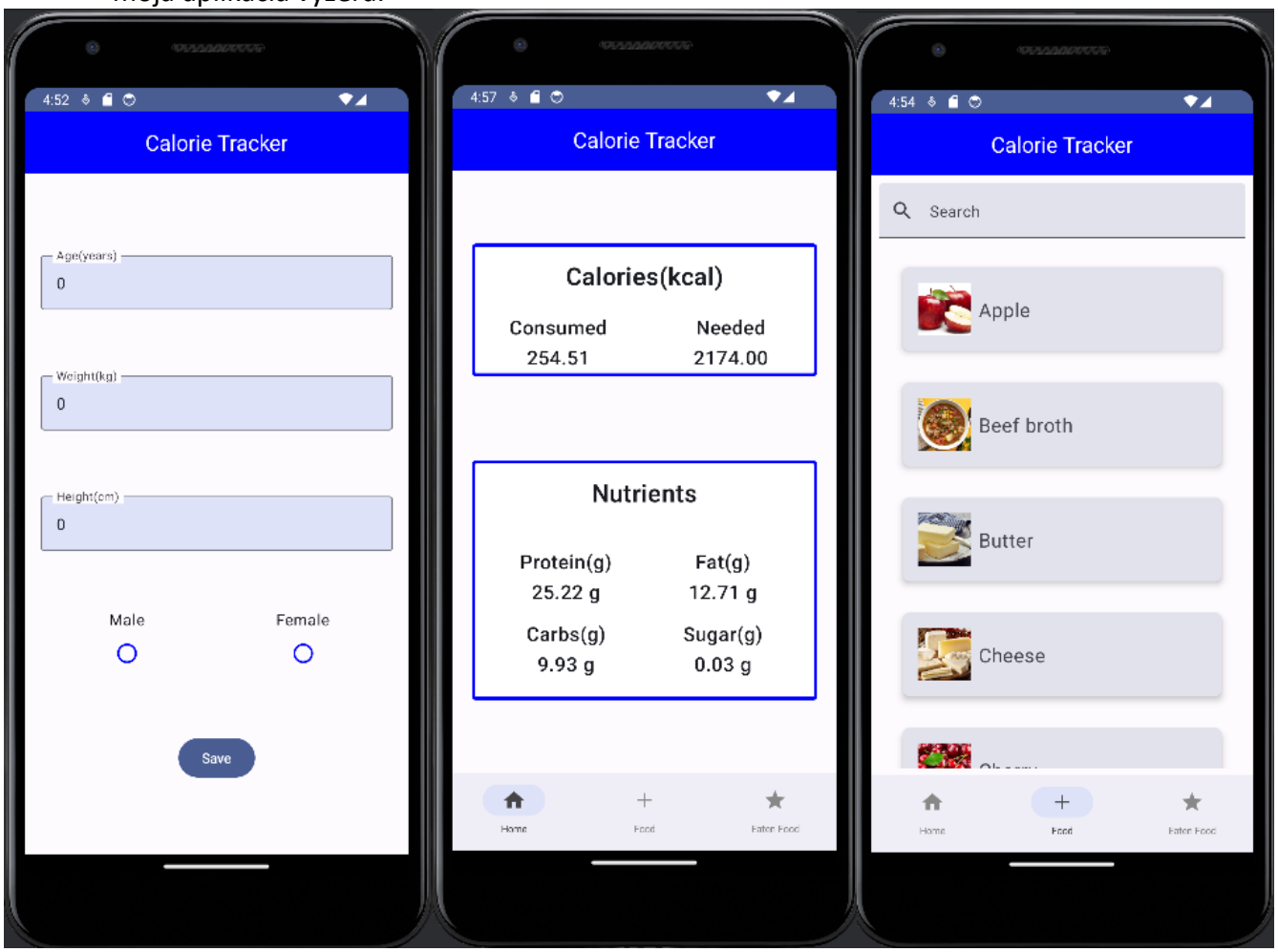
ii. Kalorické tabuľky

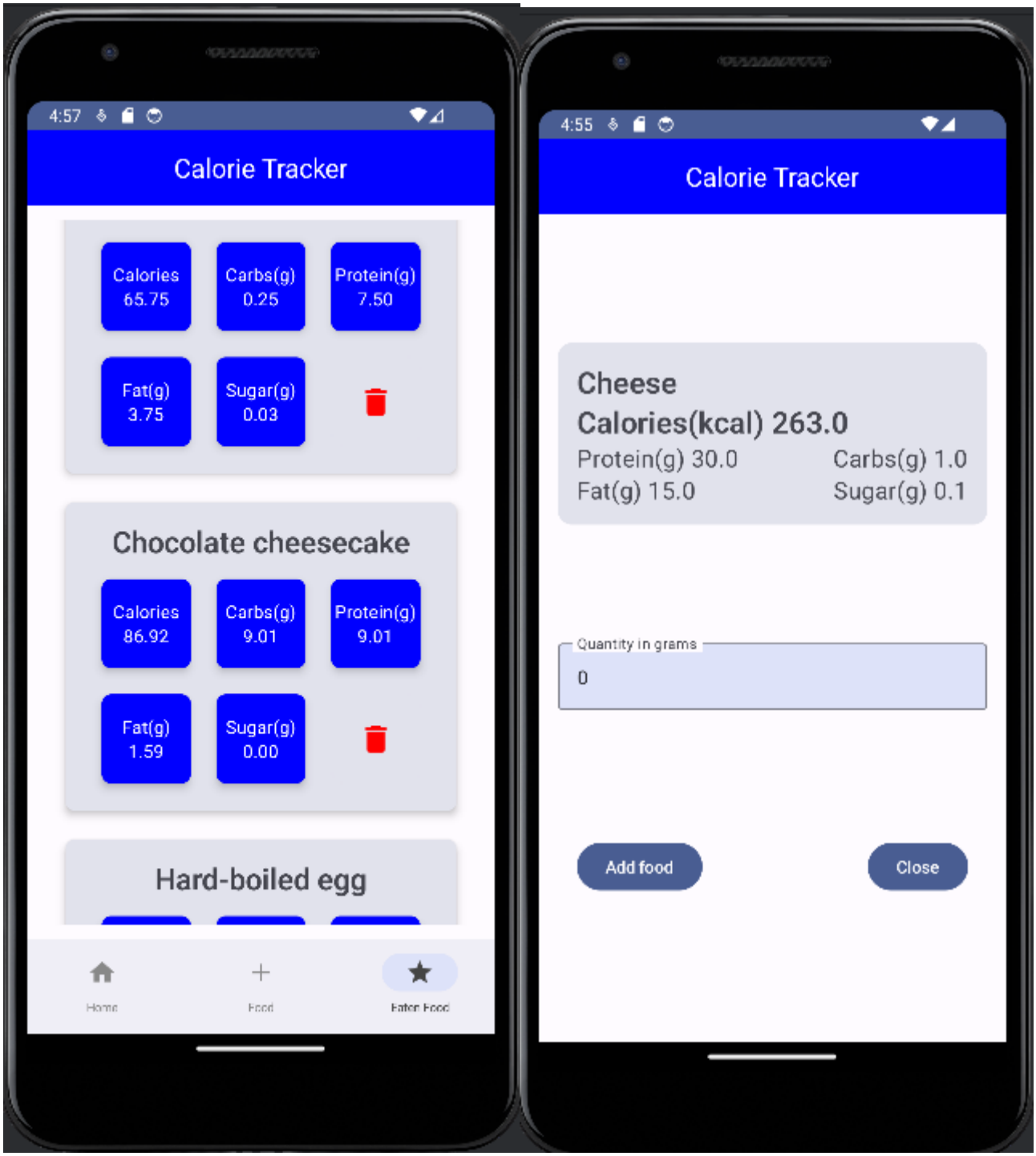
Kalorické tabuľky je nástroj, ktorý slúži na sledovanie príjmu kalórií a iných nutričných hodnôt v potravinách a nápojoch. Túto aplikáciu aj osobne používam. Táto aplikácia má rozsiahlu databázu potravín s ich kalorickým obsahom a obsahom ich makroživín (proteínov, sacharidov, tukov) a mikroživín (vitamíny, minerály), ktoré teda aplikácia umožňuje zaznamenávať.
Hlavnými funkciami tejto aplikácie je vyhľadávanie potravín. Užívatelia môžu vyhľadávať konkrétne potraviny alebo nápoje a získať informácie o ich kalorickom obsahu a nutričných hodnotách.
Ďalej je to sledovanie príjmu kalórií. Aplikácia umožňuje užívateľom sledovať ich príjem kalórií na základe konzumovaných potravín a nápojov. Toto sledovanie im pomáha udržať si kontrolu nad ich stravovacími návykmi a dosiahnuť ich zdravotné ciele, ako je chudnutie, udržiavanie hmotnosti alebo budovanie svalov.
Aplikácia umožňuje aj zaznamenávanie jedálnička. Užívatelia môžu zaznamenávať svoje jedlá a nápoje, ktorý pomáha sledovať ich stravovacie návyky a zloženie jedálnička.
Nájdu sa tu aj štatistiky, ktoré zobrazujú ich pokrok pri dosahovaní cieľov týkajúcich sa stravy a životného štýlu.



iii. Moja aplikácia

Od vyššie uvedených aplikácií sa moja aplikácia asi hlavne líši výzorom a množstvom možných možností výberu jedál. Ďalej sa moja aplikácia líši, tým že je určená iba na konkrétny deň, teda že používateľ si môže prezerať svoj denný príjem, vyššie uvedené aplikácie umožňujú pozerať aj dni predtým. Na nasledujúcich obrázkoch môžete vidieť ako moja aplikácia vyzerá.

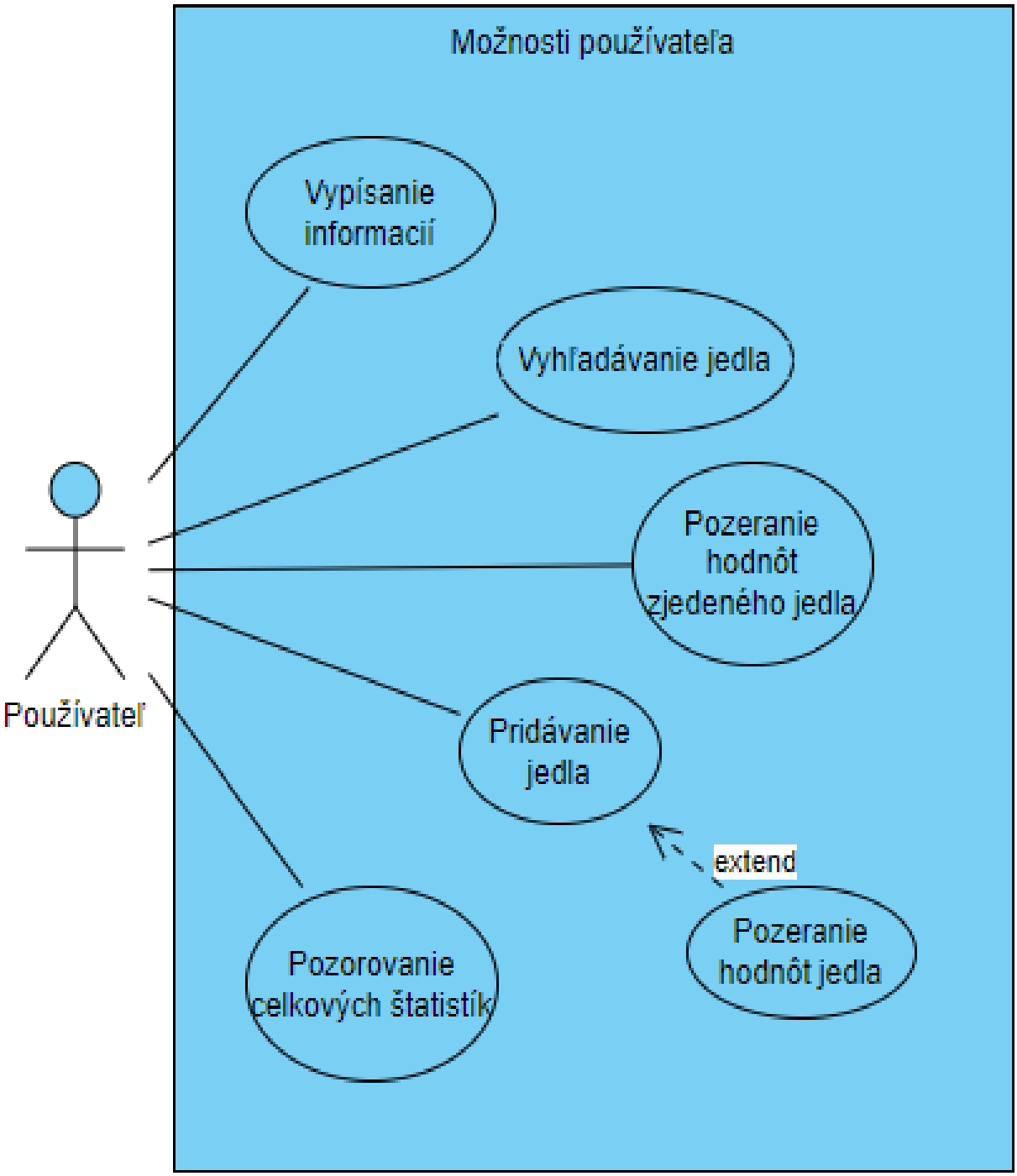



4:57
Calorie Tracker
Calories 65.75
Carbs(g) 0.25
Protein(g) 7.50
Fat(g) 3.75
Sugar(g) 0.03
Chocolate cheesecake
Calories 86.92
Carbs(g) 9.01
Protein(g) 9.01
Fat(g) 1.59
Sugar(g) 0.00
Hard-boiled egg
Home
Food
Eaten Food
4:55
Calorie Tracker
Cheese
Calories(kcal) 263.0
Protein(g) 30.0
Carbs(g) 1.0
Fat(g) 15.0
Sugar(g) 0.1
Quantity in grams
0
Add food
Close

2. Návrh riešenia problému

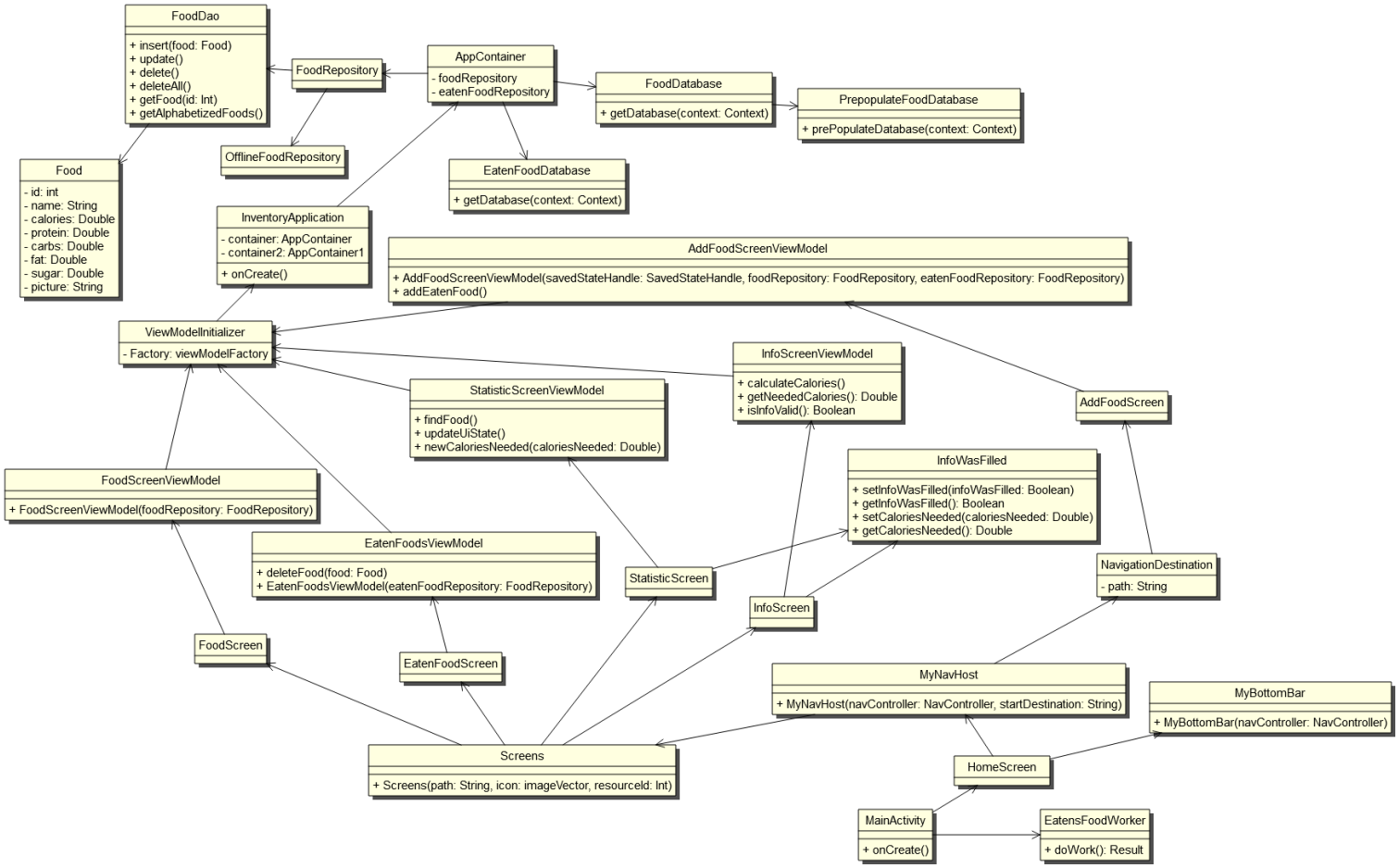
a. Krátka analýza

i. Diagram prípadov použitia



b. Návrh aplikácie

i. Diagram tried



3. Popis Implementácie

a. Databáza

Databáza v tejto aplikácií obsahuje jedlo ako svoju entitu.
Aplikácia obsahuje potom dve databázy, jednu pre všetky jedlá a jednu pre zjedené jedlá.
Obidve databázy sú potom obsiahnuté v AppContainery ku ktorému zase pristupuje InventoryApllication, z ktorej sa pri inicializáciíí viewModelov do nich posielajú potrebné databázy.

```
fun getDatabase(context: Context): FoodDatabase {
    return Instance ?: synchronized( lock: this) {
        Room.databaseBuilder(context, FoodDatabase::class.java,  name: "food_database") Builder<FoodDatabase>
            .addCallback(PrepopulateFoodDatabase(context))
            .build() FoodDatabase
            .also { it: FoodDatabase
            Instance = it
        }
    }
}
```

Obidve databázy majú potom rovnaké rozhranie funkcií, ale databáza pre všetky jedlá sa napĺňa priamo pri vzniku z json súboru.

```
suspend fun prePopulateDatabase(context: Context) {
    try {

        val foodDao = FoodDatabase.getDatabase(context).foodDao()

        val foodList: JSONArray =
            context.resources.openRawResource(R.raw.food).bufferedReader().use { it: BufferedReader
                JSONArray(it.readText())
            }

        foodList.takeIf { it.length() > 0 }?.let { list ->
            for (i in 0 ≤ until  < list.length()) {
                val food = list.getJSONObject(i)
                foodDao.insert(
                    Food(
                        name = food.getString( name: "name"),
                        calories = food.getDouble( name: "calories"),
                        protein = food.getDouble( name: "protein"),
                        carbs = food.getDouble( name: "carbs"),
                        fat = food.getDouble( name: "fat"),
                        sugar = food.getDouble( name: "sugar"),
                        picture = food.getString( name: "picture")
                    )
                )
            }
            Log.e( tag: "PrepopulateFoodDatabase",  msg: "Database prepopulated") ^let
        }
    } catch (e: Exception) {
        Log.e( tag: "PrepopulateFoodDatabase",  msg: "Error prepopulating database: ${e.message}")
    }
```

b. Navigácia

Navigácia je v aplikácií vytvorená hlavne za pomoci Bottombaru a navControlerra, ktorým sa môžete prepínať medzi obrazovkami EatenFoodScreen, FoodScreen a StatisticScreen. Zvyšné obrazovky nemajú bottomBar a navigujú sa na inú obrazovku pri stlačení tlačidla. Celá navigácia je potom obalená v tirede MyNavHost, ktorá riadi ako a kde sa treba navigovať.
Taktiež používa triedu InfoWasFilled, ktorá má v sebe uloženú informáciu či už používateľ vyplnil údaje a ak áno tak už sa obrazovka na vypĺňanie údajov nezobrazí.

```
NavHost(navController = navController,
    startDestination = startDestination
) { this: NavGraphBuilder

    composable(Screens.InfoScreen.path) { this: AnimatedContentScope  it: NavBackStackEntry
        InfoScreen(
            navigateToStatisticScreen = {
                navController.navigate(Screens.StatisticScreen.path)
            }
        )
    }

    composable(Screens.StatisticScreen.path) { this: AnimatedContentScope  it: NavBackStackEntry
        StatisticScreen()
    }
    composable(Screens.FoodScreen.path) { this: AnimatedContentScope  it: NavBackStackEntry
        FoodScreen(
            navigateToEatenFoodScreen = { it: Int
                navController.navigate( route: "${AddFoodDestination.path}/$it")
            }
        )
    }

    composable(Screens.EatenFoodsScreen.path) { this: AnimatedContentScope  it: NavBackStackEntry
        EatenFoodScreen()
    }
```

c. Obrazovky

i. AddFoodScreen

Na tejto obrazovke sa zobrazuje používateľovi jeho vybrané jedlo a informácie o ňom. Používateľ tu môže napísať aké množstvo jedla zjedol a tlačidlom ho pridať.
Toto funguje vďaka viewModelu pre túto obrazovku.
Tento si v sebe v stave drží vybrané jedlo a taktiež mu je pri vytváraní odovzdaná databáza pre zjedené jedlo.
Pomocou funkcií viewModelu sa teda pri stľačení tlačidla vloží jedlo do danej databázy a zobrazenie informácií funguje tak že si zoberieme od viewModelu dané jedlo ako stav a toto jedlo pozná svoje informácie, ktoré teda ukazujeme na displeji.

```
fun addEatenFood() {
    viewModelScope.launch { this: CoroutineScope
        val food = foodUiState.value?.food
        val quantity = quantity.value
        if (food != null && isQuantityValid()) {
            val newFood = food.copy(
                id = System.currentTimeMillis().toInt(),
                calories = String.format("%.2f", food.calories / 100 * quantity).replace( oldValue: ",", newValue: ".").toDouble(),
                protein = String.format("%.2f", food.protein / 100 * quantity).replace( oldValue: ",", newValue: ".").toDouble(),
                carbs = String.format("%.2f", food.carbs / 100 * quantity).replace( oldValue: ",", newValue: ".").toDouble(),
                fat = String.format("%.2f", food.fat / 100 * quantity).replace( oldValue: ",", newValue: ".").toDouble(),
                sugar = String.format("%.2f", food.sugar / 100 * quantity).replace( oldValue: ",", newValue: ".").toDouble()
            )
            eatenFoodRepository.insert(newFood)
        }
    }
}
```

ii. EatenFoodScreen

Táto obrazovka takisto využíva viewModel, ktorý v sebe drží zoznam zjedených jedál získaných z databázy. Táto obrazovka tento zoznam zobrazuje.
Poskytuje aj možnosť vymazať jedlo a to pomocou viewModelu, ktorý pošle databáze správu aby vymazala jedlo.

iii. FoodScreen

Táto obrazovka takisto zobrazuje všetky jedlá z databázy, ale z databázy pre všetky jedlá, ktoré si znova viewModel drží v svojom stave a získava ich z databázy.
Obrazovka navyše umožňuje vyhľadávanie medzi jedlom, ktoré je umožnené za pomoci textFieldu a funckcie filter, ktorá list jedál prefiltruje na základe toho či sa zadaný reťazec nachádza v mene jedla.

iv. InfoScreen

Táto obrazovka slúži na to aby pri prvom otvorení mohol používateľ zadať svoje údaje o veku, výške, váhe a pohlaví.
Z týchto údajov, ktoré sú znova uložené v viewModeli pomocou funkcie viewModelu vypočítajú potrebné kalórie pre používateľa.
Tie sú následne uložené vo viewModeli. Z ktorého si túto informáciu berie InfoScreen aby ho mohol poslať do triedy InfoWasFilled, ktorá si túto informáciu uchováva aj po zatvorení a opätovnom otvorení aplikácie, čo je potrebné pre poslednú spomenutú obrazovku.

v. StatisticScreen

Táto obrazovka teda zobrazuje informácie o dennom príjme používateľa.
Berie si ako som už spomínal informáciu o potrebnom množstve kalórií z triedy InfoWasFilled.
Ostatné informácie získava z databázy pre zjedené jedlá kde spočítava koľko majú dokopy kalórií a nutričných hodnôt.
Toto všetko sa deje vo viewModeli tejto obrazovky, ktorý si tieto informácie teda v sebe aj drží.
Obrazovka si potom iba tieto informácie od neho berie a zobrazuje ich.

d. WorkManager

Ako posledné by som ešte spomenul využitie workManagera. Ten je využívaný na pravidelné

vymazávanie jedál z databázy zjedených jedál.

```
class EatensFoodWorker(appContext: Context, params: WorkerParameters):
        CoroutineWorker(appContext, params) {
    Lesko
    override suspend fun doWork(): Result = coroutineScope { this: CoroutineScope
        val eatenFoodRepository = AppDataContainer2(applicationContext).eatenFoodRepository
        eatenFoodRepository.deleteAll()
        Result.success() ^coroutineScope
    }
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    val current = Calendar.getInstance()
    val midnight = Calendar.getInstance().apply { this: Calendar
        set(Calendar.HOUR_OF_DAY, 0)
        set(Calendar.MINUTE, 0)
        set(Calendar.SECOND, 0)
        add(Calendar.DAY_OF_MONTH, amount: 1)
    }
    val initialDelay = midnight.timeInMillis - current.timeInMillis

    val workRequest = PeriodicWorkRequestBuilder<EatensFoodWorker>( repeatInterval: 24, TimeUnit.HOURS)
        .setInitialDelay(initialDelay, TimeUnit.MILLISECONDS)
        .build()

    WorkManager.getInstance( context: this).enqueueUniquePeriodicWork(
        uniqueWorkName: "EatensFoodWorker",
        ExistingPeriodicWorkPolicy.KEEP,
        workRequest
    )
```

4. Zoznam použitých zdrojov

https://medium.com/@satyajeetmohalkar/pre-populating-your-app-database-using-room-using-json-file-abbc95140cc3

https://www.geeksforgeeks.org/shared-preferences-in-android-with-examples/