Dokumentácia

Popis a analýza riešeného problému

Aplikácia, ktorú som sa rozhodol spraviť ako semestrálnu prácu bude slúžiť na zaznamenávanie cvikov počas posilňovania. Bude mat v sebe ukladanie dát, ako názov daného cviku, čas v minútach koľko sa má daný cvik vykonávať, kalórie, ktoré sa pri cvičení cviku spália a krátky popis cviku. Užívateľ si môže prechádzať jednotlivé cviky a vyberať, ktorý chce vykonávať. Je možné pridať aj nový cvik po prípade starý odstrániť. Ak si užívateľ vyberie daný cvik môže si prečítať krátky popis k cviku a ak sa rozhodne ho cvičiť spustí časovač. Časovač je možné kedykoľvek zastaviť znovu pustiť a zresetovať. Súčasťou aplikácie je aj počítač krokov, ktorý po spustení začne počítať kroky užívateľovi. Taktiež sa dá počítanie zastaviť a znovu spustiť.

Spracovanie prehľadu podobných aplikácií

Podobných aplikácií je dostupných naozaj veľa. Niektoré sú jednoduchšie tie väčšinou bývajú populárnejšie a mávajú lepšie hodnotenia, keďže človek chce mať aplikáciu ako je táto čo najprehľadnejšia. Podobné appky sú napríklad RepCount Gym Workout Tracker, Strong Worukout Tracker Gym Log, Alpha Progression Gym Logger a mnoho davších podobných.

Rozdiely medzi podobnými aplikáciami a našou je v tom že neobsahujú počítanie krokov, taktiež vo väčšine prípadov sa cviky pridávať nedajú alebo idú pridať len cviky, ktoré sú vopred zvolené teda nie je možné vytvoriť si tréning úplne podľa seba. Avšak väčšinou obsahujú nejaký systém pre zaznamenávanie hmotnosti aby si užívateľ mohol sledovať progres. Taktiež užívateľské rozhranie je prepracovanejšie a lákavejšie. Ale na druhú stranu naša aplikácia je zase jednoduchá prehľadná a jasná čo môže byt tiež plus u niektorých ľudí.

Návrh riešenia problému

Pri riešení implementácie časti krokomeru je potrebné zapojiť senzor zo zariadenia pokiaľ ho zariadenie obsahuje. Taktiež je potrebné implementovať zastavenie merania pokiaľ to užívateľ vyžaduje.

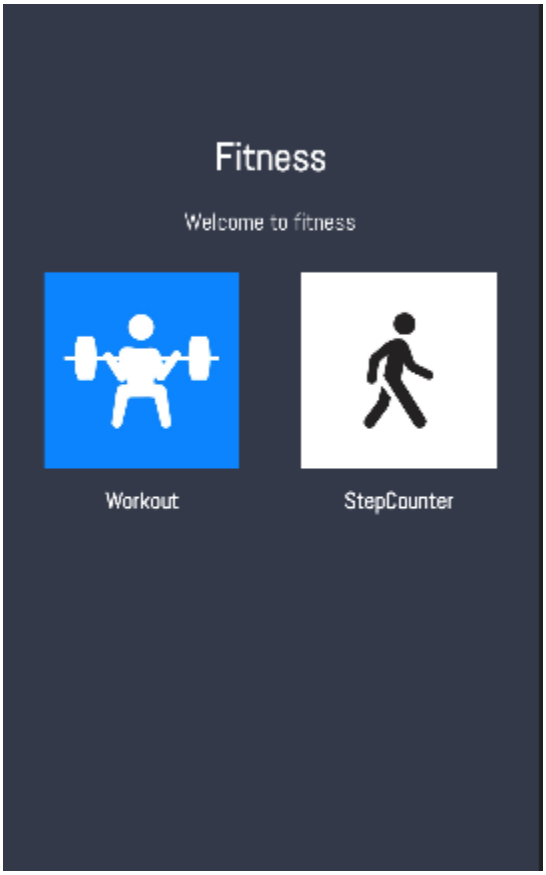
Druhá časť obsahujúca cviky musí byť riešená skrz databázu teda cviky sa musia vypisovať z databázy a nesmú sa stratiť záznamy po zatvorení aplikácie. Je nutné definovanie funkcie pridania a odobratia cviku ako aj detailný prehľad cviku.

V detailnom pohľade je nutné implementovanie časomeru pre dopovedajúci cvik teda čas sa musí meniť s ohľadom na daný cvik je nutne vedieť časomieru zastaviť znovu pustiť po prípade zresetovať.

Implementácia

Začnime implementáciou hlavnej obrazovky aplikácie. Tejto obrazovke zodpovedá aktivita MainAktivity ktorá je v celku jednoduchá nastavujú sa tu pri vytvorení dva onClickListner-y kde jeden spúšťa aktivitu pre zoznam cvičení a druhý akivitu pre krokomer. Layout je jednoduchý

obsahuje meno krátke privítanie aplikácie a dve LinearLayouty s obrazkami a textovým popisom. Layout fungue aj pri otočení obrazovky na stranu.



Krokomer

Po kliknutí na jeden z obrázkov sa spustí príslušná aktivita. Po kliknutí na StepCounter sa Spustí príslušná aktivita kde na začiatku sa deklarujú premenne pre napočítavanie krokov, ďalej premenná boolean, ktorá symbolizuje či užívateľ chce kroky zaznamenávať alebo nie a hlavne premenná reprezentujúca senzor. Ďalšími sú premenné zodpovedajúce častiam layoutu, ktoré sa majú počas aktivity meniť. Taktiež táto aktivita implementuje rozhranie SensorEventListener a teda aj jeho dve povinne metódy onSensorChanged kde sa definuje čo sa má diať pokiaľ senzor zaznamená zmenu :

```
override fun onSensorChanged(event: SensorEvent) {
    if (running && event.sensor.type == Sensor.TYPE_STEP_DETECTOR) {
        steps++
        stepTV.text = steps.toString()
    }
}
```

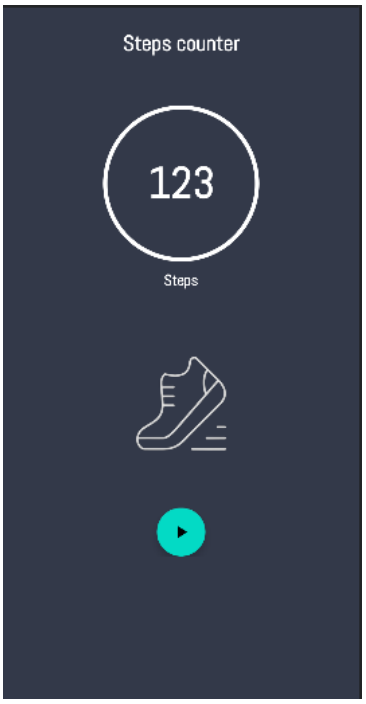
a druhou je onAccuracyChange, ktorú ale nie je nutné v našom prípade implementovať

```
override fun onAccuracyChanged(p0: Sensor?, p1: Int) {

}
```

V onCreate funkcii sú definované príslušné premenné, ktoré boli spomenuté a taktiež nastavené onClickListner na tlačidlo z layoutu zodpovedajúce spusteniu a zastaveniu počítania krokov. Toto tlačidlo v sebe využíva metódu startCounting, ktorá spustí počítanie pokiaľ zariadenie obsahuje daný typ senzoru.

```
private fun startCounting() {
    running = true
    val sensor = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_DETECTOR)
    if (sensor != null) {
        sensorManager.registerListener( listener: this, sensor, SensorManager.SENSOR_DELAY_NORMAL)
    } else {
        Toast.makeText( context: this,  text: "Step detector sensor not found", Toast.LENGTH_SHORT).show()
    Andrej-Dusa
    }
}
```

Layout pre krokmer:



Zoznam cvikov

Pokiaľ užívateľ zaklikne druhú možnosť pre cvičenia spustí sa príslušná aktivita WorkoutActivity, kde je využitý Data Binding na editovanie častí layoutu. Taktiež sa tu využíva Room databáza na vypísanie zoznamu cvikov do RecyclerView v layoute.

```
private val workoutDB : WorkoutDatabase by lazy {
    Room.databaseBuilder( context: this,WorkoutDatabase::class.java,  name: "workoutsDTB")
        .allowMainThreadQueries()
        .fallbackToDestructiveMigration()
        .build()
}
```

```
private fun setupRecyclerView(){
    binding.idWorkoutRV.apply { this: RecyclerView
        layoutManager=LinearLayoutManager( context: this@WorkoutAkitivity)
        adapter=workoutAdapter
    }

}
```

Na toto sa využíva pomocná trieda WorkoutAdapter. Kde na to slúži vnútorná trieda aj s jej metódou:

```
inner class ViewHolder : RecyclerView.ViewHolder(binding.root) {
    Andrej-Dusa *
    @SuppressLint("SetTextI18n")
    fun bind(item: Workout) {
        binding.apply { this: ItemBinding
            idTVActivity.text = item.name
            idTVWorkoutTime.text = item.time.toString() + " min"
            idIVItem.setImageResource(item.img)

            root.setOnClickListener { it: View!
                val intent=Intent(context,WorkoutDetailActivity::class.java)
                intent.putExtra( name: "bundle_workout_id", item.id)
                context.startActivity(intent)
            }

        }
    }
}
```

Taktiež v aktivite pre cviky je vďaka Data Bindingu spravené editovanie polí pre celkové spálené kalórie pri tréningu a celkový čas tréningu. Taktiež aktualizácia zoznamu cvikov cez preťaženú metódu onResume:

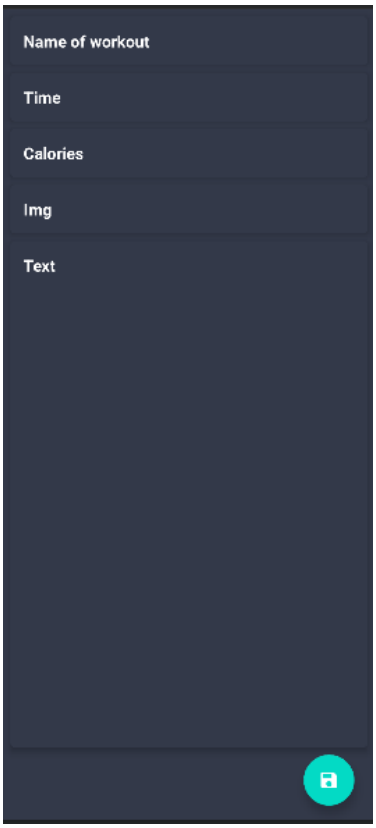
```
override fun onResume() {
    super.onResume()
    checkItem()
}
```

Kde sa prehľadávajú položky v databáze skrz metódu checkItem:

```
private fun checkItem(){
    binding.apply { this: ActivityWorkoutAkitivityBinding
        workoutAdapter.differ.submitList(workoutDB.dao().getAllWorkouts())
        setupRecyclerView()
        totalMin = 0
        totalCal = 0
        workoutDB.dao().getAllWorkouts().forEach { it: Workout
            totalMin += it.time
            totalCal += it.calories
        }
        idTVCalories.text = totalCal.toString()
        idTVTime.text = totalMin.toString()
    }
}
```

Pridanie cviku

Obsahuje aj tlačidlo pre pridanie cviku, ktoré otvorí príslušnú aktivitu pre pridanie cviku. Ktorej zodpovedá krátky formulár.



Kde sa využíva prístup do Room databázy aj s Data Binding-om na prístup k textu z editovaných polí z formulára. Nastavuje sa tu pridanie do databázy aj s overením údajov po stlačení tlačidla uloženia.

```
binding.apply { this: ActivityAddWorkoutBinding
    btnSave.setOnClickListener { it: View!
        val name = edtTitle.text.toString()
        val time = timeEdit.text.toString()
        val calories = caloriesEdit.text.toString()
        val desc = textText.text.toString()

        //overenie ci nie su udaje vkladene do databzy prazdne
        if (name.isNotEmpty() || time.isNotEmpty() || calories.isNotEmpty()){
            workoutEntity= Workout( id: 0,name,desc,time.toInt(), calories.toInt(), R.drawable.workout)
            workoutDB.dao().insertWorkout(workoutEntity)
            finish()
        }
        else{
            Snackbar.make(it, text: "Name, time and calories must be filled!",Snackbar.LENGTH_LONG).show()
        }
    }
}
```

Detailný cvik

Po zakliknutí jedného z cvikov z RecyclerView sa nám spustí detailna aktivita cviku. Kde je opäť využitá Room databáza. Ale obsahuje premenné aj pre zaznamenávanie koľko ešte ostáva času, premenná pre záznam pôvodného času, premenná typu CountdownTimer a premenná pre zaznamenávanie či užívateľ chce aby čas ubiehal. Metódy pre kontrolu nad časomierou resetTimer, pauseTimer, startTimer:

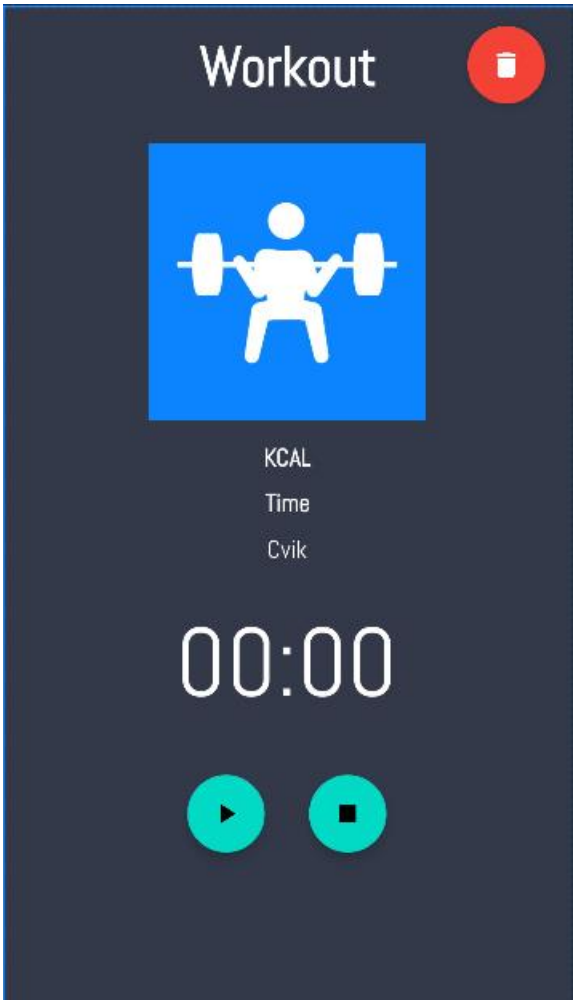
```
//reset casomiery
Andrej-Dusa
private fun resetTimer() {
    countDownTimer.cancel()
    mTimerRunning = false
    binding.idFAB1.setImageResource(R.drawable.ic_play)
    timeLeft = timeStart
    updateClock()
}

//zastavenie casomiery
Andrej-Dusa
private fun pauseTimer() {
    countDownTimer.cancel()
    mTimerRunning = false
    binding.idFAB1.setImageResource(R.drawable.ic_play)
}
```

```
private fun startTimer() {
    //inicializacia CountDowTimeru na znizovanie o 1 sekundu
    countDownTimer = object : CountDownTimer(timeLeft, countDownInterval: 1000) {
        override fun onTick(millisUntilFinished: Long) {
            timeLeft = millisUntilFinished
            updateClock()
        }

        //co sa ma stat ak cas uplynie
        override fun onFinish() {
            mTimerRunning = false
            binding.idTVClock.text = "Done!"
        }
    }.start()
    mTimerRunning = true
    binding.idFAB1.setImageResource(R.drawable.ic_pause)
}
```

Metóda pre správne vypisovanie času:

```
//Update textView casomeru
Andrej-Dusa
private fun updateClock() {
    val minutes: Int = ((timeLeft / 1000)/60).toInt()
    val seconds: Int = ((timeLeft / 1000)%60).toInt()

    val timeLeftString = String.format(Locale.getDefault(), format: "%02d:%02d", minutes, seconds)
    binding.idTVClock.text = timeLeftString
}
```

Layout pre príslušný cvik:



Databáza

Pre správny chod našej aplikácie bolo nutné implementovať jednoduchú Room databázu. Zodpovedajú jej dátova trieda Workout, ktorá v sebe obsahuje entitu tabuľky.

```
@Entity(tableName = "workouts")
data class Workout(
    @PrimaryKey(autoGenerate = true)
    val id:Int,
    @ColumnInfo(name = "name")
    val name:String,
    @ColumnInfo(name = "text")
    val text: String,
    @ColumnInfo(name = "time")
    val time: Int,
    @ColumnInfo(name = "calories")
    val calories: Int,
    @ColumnInfo(name = "img")
    val img: Int
)
```

Abstraktnú triedu pre datbázu WorkoutDatabase:

```
@Database(entities = [Workout::class], version = 1)
abstract class WorkoutDatabase : RoomDatabase(){
    Andrej-Dusa
    abstract fun dao(): WorkoutDao
}
```

A rozhrania WorkoutDao kde sú implementované všetky možne operácie and tabuľkou:

```
@Insert(onConflict = OnConflictStrategy.REPLACE)
fun insertWorkout(noteEntity: Workout)
```

```
@Delete
fun deleteWorkout(noteEntity: Workout)
```

```
@Query("SELECT * FROM 'workouts' ORDER BY id DESC")
fun getAllWorkouts() : MutableList<Workout>
```

```
@Query("SELECT * FROM 'workouts' WHERE id LIKE :id")
fun getWorkout(id : Int) : Workout
```