Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Vývoj mobilnej aplikácie pre iOS

Systémová príručka

2022 **Martin Varga**

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Vývoj mobilnej aplikácie pre iOS

Systémová príručka

Študijný program:	Inteligentné systémy
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra kybernetiky a umelej inteligencie (KKUI)
Školiteľ:	Ing. Erik Kajáti, PhD.

Košice 2022 **Martin Varga**

Obsah

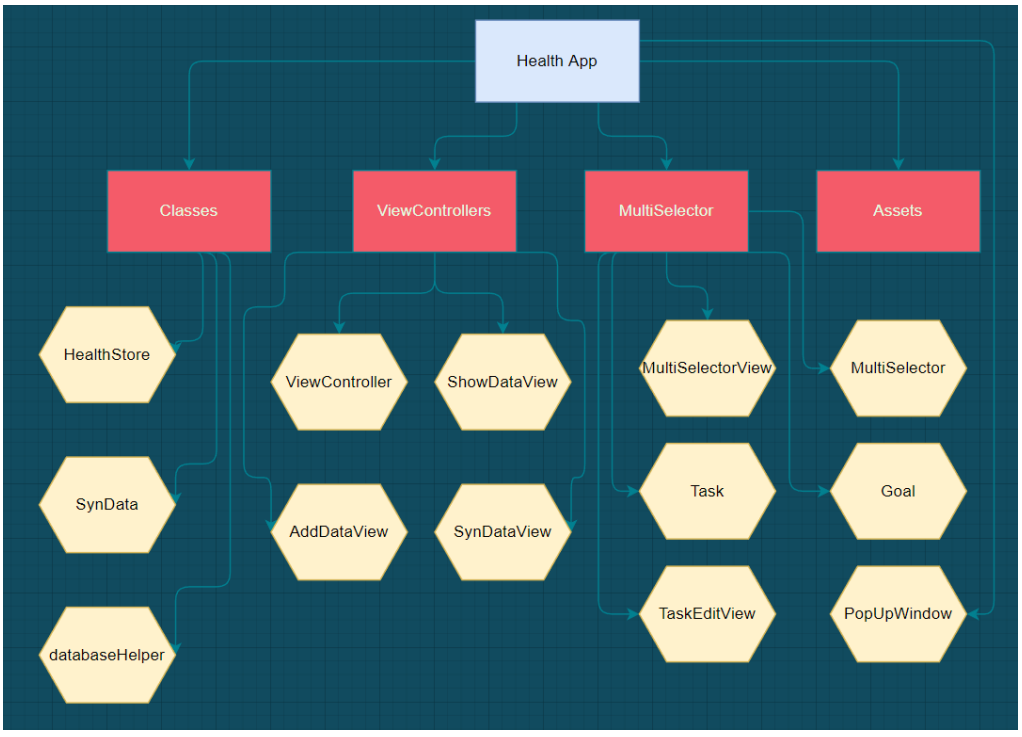
1 Potrebný softvér a knižnice

V tejto časti vymenujem a v krátkosti popíšem potrebný softvér pre vývoj aplikácie a použité knižnice.

1.1 Xcode

Xcode je vývojové prostredie od spoločnosti Apple. Pri našom riešení bakalárskej práce bolo toto vývojové prostredie použité na vývoj aplikácie pomocou programovacieho jazyku Swift, ale Xcode samozrejme podporuje aj plno ďalších programovacích jazykov. Xcode ponúka množstvo funkcionalít, ktoré vývojár ocení a práca v ňom je veľmi intuitívna.

1.1.1 Štruktúra projektu v Xcode



Obr. 1.1: Diagram projektu

Na Obr. 1.1 môžeme vidieť diagram projektu. Všetky súbory a kód, ktorý obsahujú popíšem v kapitole 2.

1.2 Použité knižnice

1.2.1 HealthKit

HealthKit umožňuje aplikácií pracovať so zdravotnými dátami cez *HKHealthStore* objekt. Pomocou HealthKitu vieme zapisovať, čítať a analyzovať rôzne zdravotné dáta pomocou dopytov na už spomenutý objekt. Dopyty môžu byť jednoduché, ale taktiež s použitím rôznych voliteľných modifikátorov sa môžu stať veľmi komplikovanými.

1.2.2 SwiftUI

SwiftUI je knižnica, ktorá vývojárovi umožňuje vyvíjať užívateľské rozhranie pre aplikáciu. Táto knižnica je jednou z najmodernejších pre vývoj užívateľského rozhrania, keďže je relatívne nová. Apple často pridáva nové funkcionality do tejto knižnice a vízia Applu je taká, že sa na túto knižnicu postupne prejde zo staršieho UIKit.

1.2.3 UIKit

Tak isto ako SwiftUI, tak aj UIKit je knižnica od Applu pre vývoj užívateľského rozhrania. Existuje už dlhšiu dobu, takže väčšinu potrebnej funkcionality pre vývoj užívateľského rozhrania už v sebe má. Apple aj napriek svojmu zameraniu na SwiftUI stále podporuje a prináša aktualizácie aj pre UIKit. Najväčšou výhodou UIKit je *Storyboard*, kde vývojár dokáže všetky elementy užívateľského rozhrania pridať do pohľadu.

1.2.4 SQLite3

SQLite3 je knižnica pomocou ktorej dokážeme implementovať metódy na vytvorenie a komunikáciu s SQLite databázou. Taktiež je možné implementovať posielanie rôznych SQL dopytov a všetko ostatné, čo ponúkajú SQL databázy.

1.2.5 Foundation

Foundation je knižnica, ktorá je importovaná automaticky pri vytvorení nového *.swift* súboru. Ponúka základnú vrstvu funkcionality pre aplikáciu a ostatné knižnice v rámci aplikácie.

2 Kód aplikácie

2.1 HealthStore

2.1.1 Štruktúra healthModelSample

Zdrojový kód 2.1: healthModelSample

```
struct healthModelSample{
    var uuid : String
    var sampleType : String
    var sampleValue : Double
    var startDate : Date
    var endDate : Date
}
```

Podľa štruktúry zobrazenej v ukážke kódu 2.1 vytvárame premenné, ktoré zodpovedajú tomuto štruktúrovanému typu. Tie sa využívajú na synchronizáciu s lokálnou databázou.

2.1.2 Premenné

Zdrojový kód 2.2: Premenné

```
var healthStore: HKHealthStore?
var waterLabel: String?
var stepLabel: String?
var distanceRunningLabel: String?
var distanceCycledLabel: String?
var distanceSwimmingLabel: String?
var distanceWheelchairLabel: String?
var heartRateLabel: String?
var caloriesLabel: String?
```

```
var samplesToSyncWater:[healthModelSample]
= [healthModelSample]()
var samplesToSyncSteps:[healthModelSample]
= [healthModelSample]()
var samplesToSyncDistanceRunning:[healthModelSample]
= [healthModelSample]()
var samplesToSyncDistanceSwimming:[healthModelSample]
= [healthModelSample]()
var samplesToSyncDistanceCycling:[healthModelSample]
= [healthModelSample]()
var samplesToSyncDistanceWheelchair:[healthModelSample]
= [healthModelSample]()
var samplesToSyncHeartRate:[healthModelSample]
= [healthModelSample]()
var samplesToSyncCalories:[healthModelSample]
= [healthModelSample]()
```

V ukážke kódu 2.2 môžeme vidieť všetky premenné, ktoré patria triede HealthStore. Jedná sa o premenné do ktorých sa zapíše premenná po dopyte na *HKHealthStore* objekt. Každá premenná je rezervovaná pre iný dopyt. V druhej časti sú polia, do ktorých sa budú pridávať konkrétne záznamy premenných podľa vyššie definovanej štruktúry a rovnako ako premenné vyššie tiež je každé pole rezervované pre iný dopyt. Okrem týchto premenných tu je aj *HKHealthStore* objekt s ktorým budeme pracovať pri dopytoch o zdravotné dáta.

2.1.3 HealthStore singleton

Zdrojový kód 2.3: HealthStore singleton

```
static let shared = HealthStore()

private init() {
    if HKHealthStore.isHealthDataAvailable() {
        healthStore = HKHealthStore()
    }
}
```

Táto časť kódu zabezpečí, že vždy existuje len jedna inštancia triedy HealthStore, keďže budeme pre našu aplikáciu vždy potrebovať len jednu inštanciu tohto

objektu. Využíva návrhový vzor singleton.

2.1.4 requestAuthorization

Zdrojový kód 2.4: requestAuthorization

```
func requestAuthorization(completion:
@escaping(Bool)->Void){
let stepType=HKQuantityType.quantityType(
forIdentifier:HKQuantityTypeIdentifier.stepCount)!
let dateOfBirth=HKObjectType.characteristicType(
forIdentifier:.dateOfBirth)!
let bloodType=HKObjectType.characteristicType(
forIdentifier:.bloodType)!
let biologicalSex=HKObjectType.characteristicType(
forIdentifier:.biologicalSex)!
let bodyMassIndex=HKObjectType.quantityType(
forIdentifier: .bodyMassIndex)!
let height=HKObjectType.quantityType(
forIdentifier:.height)!
let bodyMass=HKObjectType.quantityType(
forIdentifier:.bodyMass)!
let activeEnergy=HKObjectType.quantityType(
forIdentifier:.activeEnergyBurned)!
let waterType=HKObjectType.quantityType(
forIdentifier:HKQuantityTypeIdentifier.dietaryWater)!
let distanceRanType=HKObjectType.quantityType(
forIdentifier:HKQuantityTypeIdentifier
.distanceWalkingRunning)!
let distanceCyclingType=HKObjectType.quantityType(
forIdentifier:HKQuantityTypeIdentifier
.distanceCycling)!
let distanceSwimmingType=HKObjectType.quantityType(
forIdentifier:HKQuantityTypeIdentifier
.distanceSwimming)!
let distanceWheelchairType=HKObjectType.quantityType(
forIdentifier:HKQuantityTypeIdentifier
.distanceWheelchair)!
```

```
let heartRateType=HKObjectType.quantityType(
forIdentifier:HKQuantityTypeIdentifier.heartRate)!
let caloriesType=HKObjectType.quantityType(
forIdentifier:HKQuantityTypeIdentifier
.dietaryEnergyConsumed)!
let healthKitTypesToWrite: Set<HKSampleType> =
    [stepType,
    bodyMassIndex,
    activeEnergy,
    waterType,
    distanceRanType,
    distanceCyclingType,
    distanceSwimmingType,
    distanceWheelchairType,
    heartRateType,
    caloriesType,
    HKObjectType.workoutType()]
let healthKitTypesToRead: Set<HKObjectType> =
    [stepType,
    dateOfBirth,
    bloodType,
    biologicalSex,
    bodyMassIndex,
    height,
    bodyMass,
    waterType,
    distanceRanType,
    distanceCyclingType,
    distanceSwimmingType,
    distanceWheelchairType,
    heartRateType,
    caloriesType,
    HKObjectType.workoutType()]
guard let healthStore = self.healthStore else {
return completion(false) }
healthStore.requestAuthorization(
toShare: healthKitTypesToWrite,
```

```
read: healthKitTypesToRead) { (success, error) in
    completion(success)
}
}
```

V tejto metóde zadefinujeme typy zdravotných dát. Následne typy, ktoré chceme čítať dáme do setu s názvom *healthKitTypesToRead* a typy, ktoré chceme zapisovať dáme do setu s názvom *healthKitTypesToWrite*. Následne skontrolujeme, či existuje *HKHealthStore* objekt a ak áno, tak na neho zavoláme metódu definovanú v rámci HealthKit knižnice, kde ako vstupné parametre zadáme sety spomenuté vyššie.

2.1.5 readWater

Zdrojový kód 2.5: readWater

```
func readWater(dateStart: Date, dateEnd: Date,
completion: @escaping(Bool) -> Void){
guard let waterType = HKSampleType.quantityType(
forIdentifier: .dietaryWater) else {
    print("Sample type not available")
    return
}
let s24hPredicate = HKQuery.predicateForSamples(
withStart: dateStart, end: dateEnd, options:
.strictEndDate)
let waterQuery = HKSampleQuery(sampleType: waterType,
   predicate: s24hPredicate,
   limit: HKObjectQueryNoLimit,
   sortDescriptors: nil) {
    (query, samples, error) in
        guard error == nil,
            let quantitySamples = samples as?
            [HKQuantitySample] else {
                print("Something went wrong:
                \(String(describing: error))")
                return
        }
        let total = quantitySamples.reduce(0.0) {
        $0 + $1.quantity.doubleValue(for:
```

```
        HKUnit.literUnit(with: .milli)) }
                print("total water: \(total)")
            DispatchQueue.main.async {
                self.waterLabel = "\(total) ml of water"
                if(self.waterLabel != ""){
                    self.waterLabel =
                    "\(total) ml of water"
                    completion(true)
                }else{
                    self.waterLabel = "0 ml of water"
                    completion(false)
                }
        }
    }
self.healthStore?.execute(waterQuery)
}
```

Táto metóda slúži na získanie hodnoty prijatých tekutín v rámci určitého obdobia. Ako vstupné parametre sú dátumové objekty, ktoré v sebe majú informácie o dátume s časom. Slúžia na určenie časového rozsahu od začiatku až po koniec čítanej hodnoty. Využívame tu *completion handler* na to, aby sme vedeli, kedy sa daný dopyt skončí a aby sme po skončení dopytu mohli aktualizovať hodnotu zobrazenú užívateľovi, je to vlastne taká pomôcka. *Completion handler* potrebujeme preto, lebo dopyt beží asynchrónne na inom vlákne ako hlavná časť aplikácie a bez neho by sme nevedeli presne, kedy sa dopyt dokončí. Následne definujeme typ aký chceme čítať a časový rozsah z akého nám to vráti dané záznamy, tak aby tomu dopyt rozumel. Tieto údaje využijeme pri vytváraní dopytu, následne určíme čo sa má stať s vrátenými záznamami. V tomto prípade spočítame hodnotu zo všetkých záznamov a hodnotu prevedieme na mililitre. Túto hodnotu zapíšeme už do vyššie spomenutej rezervovanej premennej. Na konci len tento dopyt vykonáme na spomenutom *HKHealthStore* objekte. V triede HealthStore je vytvorených plno podobných metód, len s malými úpravami ohľadom toho, aký typ hodnoty chceme čítať a v akých jednotkách.

2.1.6 writeWater

Zdrojový kód 2.6: writeWater

```
func writeWater(waterConsumed: Double,
```

```
dateStart: Date, dateEnd: Date, completion:
@escaping(Bool) -> Void) {
guard let waterType = HKSampleType.quantityType(
forIdentifier: .dietaryWater) else {
    print("Sample type not available")
    return
}
let waterQuantity=HKQuantity(unit:HKUnit
.literUnit(with:.milli),doubleValue:waterConsumed)
let waterQuantitySample=HKQuantitySample(type:waterType,
quantity:waterQuantity,start:dateStart,end:dateEnd)
self.healthStore?.save(waterQuantitySample)
{(success, error) in
    if (error != nil) {
      NSLog("error occurred saving water data")
        completion(false)
    }else{
        completion(true)
    }
}
}
```

V tejto metóde podobne ako v predchádzajúcej vytvárame dopyt na *HKHealthStore* objekt. Taktiež tu využívame *completion handler* na to, aby sme vedeli, kedy sa dopyt dokončil. Rozdiel spočíva v tom, že v tomto prípade chceme pridať novú hodnotu do zariadenia, takže ako vstupný parameter tu pribudla hodnota, ktorú chceme pridať. Tu konkrétne sa jedná o *Double* premennú, ktorá predstavuje hodnotu v mililitroch. Určíme typ, časový rozsah a pomocou týchto premenných vytvoríme *HKQuantitySample*, ktorý pomocou metódy *.save()* uložíme do zariadenia a takýmto spôsobom pridáme nový záznam. V knižnici HealthKit je metóda *.save()* definovaná a len ju zavoláme na *HKHealthStore* objekt, kde ako vstupný parameter dáme spomenutý nami vytvorený *HKQuantitySample*. V rámci HealthStore triedy je taktiež vytvorených veľa ďalších podobných metód, kde sa len mení typ pridávanej hodnoty.

2.1.7 readWaterForDB

Zdrojový kód 2.7: readWaterForDB

```
func readWaterForDB(dateStart: Date,
dateEnd: Date, completion:
@escaping(Bool) -> Void) {
let sampleType = HKSampleType.quantityType(
forIdentifier: HKQuantityTypeIdentifier.dietaryWater)
let predicateTime = HKQuery.predicateForSamples(
withStart:dateStart,end:dateEnd,options:.strictEndDate)
let query = HKSampleQuery.init(
    sampleType: sampleType!,
    predicate: predicateTime,
    limit: HKObjectQueryNoLimit,
    sortDescriptors: nil) { (query, results, error) in
    guard let results = results else{
        completion(false)
        return
    }
    self.samplesToSyncWater.removeAll()
    var index = 0
    for result in results{
        let sampleValues = results as? [HKQuantitySample]
        let valueToAdd = sampleValues?[index].
        quantity.doubleValue(for:
        HKUnit.literUnit(with: .milli)) ?? 69

        self.samplesToSyncWater.append(
        healthModelSample(uuid: result.uuid.uuidString,
        sampleType: result.sampleType.identifier,
        sampleValue: valueToAdd, startDate:
        result.startDate, endDate:
        result.endDate))
        index = index + 1
    }
    completion(true)
    return
}
healthStore?.execute(query)
}
```

Táto metóda je vlastne takou pomocnou metódou pre synchronizáciu. Je podobná metóde *readWater* popísanej vyššie, ale s tým rozdielom, že hodnotu nijako nespočítavame, ale každý jeden vrátený záznam z dopytu upravíme podľa potreby nami definovanej štruktúry a takýto záznam zodpovedajúci štruktúre *healthModelSample* pridáme do rezervovaného poľa, ako bolo spomenuté pri opísaní premenných tejto triedy. Predtým, ako dopyt vykonáme pole úplne pre istotu vyčistíme metódou *.removeAll()*. Takýchto pomocných metód je v tejto triede viacero a zase len s malými úpravami ohľadom toho, o aký typ zdravotných dát sa jedná.

2.2 SynData

Zdrojový kód 2.8: Trieda SynData

```
import Foundation
class SynData{
let goalNone: [Goal]
var task: Task
var selectedGoals: Set<Goal>
static let shared = SynData()
init(){
    self.goalNone = [Goal(name: "None")]
    self.task=Task(name:"",servingGoals:[goalNone[0]])
    self.selectedGoals = Set(arrayLiteral: goalNone[0])
}
func printVars(){
    print(goalNone)
    print(task)
    print(selectedGoals)
    //print(count)
    //count = count + 1
}
}
```

Táto trieda je taktiež implementovaná ako *singleton*, keďže chceme aby počas behu aplikácie vždy existovala len jedna jediná inštancia tejto triedy. V tejto triede je vlastne len uložená informácia o tom, ktoré zdravotné dáta majú byť synchronizované podľa výberu užívateľa. Žiadne metódy na prácu s týmito premennými v tejto triede neexistujú, ale menia sa priamo. Je tu implementovaná metóda na

výpis premenných v tejto triede, ale táto metóda slúžila len na debugovacie účely.

2.3 databaseHelper

V tejto triede sú implementované metódy na prácu s SQLite databázou.

createDB

Zdrojový kód 2.9: createDB

```
func createDB() -> OpaquePointer? {
let filePath = try? FileManager.default.url(for:
.documentDirectory, in: .localDomainMask,
appropriateFor: nil, create: true)
.appendingPathExtension(path)
var db : OpaquePointer? = nil
if sqlite3_open(filePath?.path, &db) != SQLITE_OK {
    print("There is error in creating DB")
    return nil
}else {
    print("Database has been created with path \(path)")
    return db
}
}
```

Táto metóda slúži na vytvorenie lokálnej databázy. Pomocou *FileManager* špecifikujeme, kde sa databáza má vytvoriť s akou príponou. V tomto konkrétnom prípade sa vytvorí v priečinku pre dokumenty danej aplikácie.

2.3.1 createTable

Zdrojový kód 2.10: createTable

```
func createTable()  {
let query = "CREATE TABLE IF NOT EXISTS health_db9
(uuid TEXT PRIMARY KEY, sampleType TEXT,
sampleValue DOUBLE, startDate TEXT,
endDate TEXT);"
var statement : OpaquePointer? = nil
if sqlite3_prepare_v2(self.db, query, -1,
```

```
&statement, nil) == SQLITE_OK {
    if sqlite3_step(statement) == SQLITE_DONE {
        print("Table creation success")
    }else {
        print("Table creation fail")
    }
} else {
    print("Prepration fail")
}
}
```

Táto metóda pomocou jednoduchého SQL dopytu vytvorí tabuľku v danej databáze za predpokladu, že tabuľka už neexistuje. V dopyte je špecifikované, ako majú vyzerať stĺpce v tabuľke a tie sa zhodujú s tým, ako sme navrhli štruktúru *healthModelSample*.

2.3.2 insert

Zdrojový kód 2.11: insert

```
func insert(uuid : String, sampleType : String,
sampleValue: Double, startDate: Date, endDate: Date){
let formatter = DateFormatter()
formatter.dateFormat = "dd-MM-yyyy HH:mm:ss"
let dateStartString = formatter.string(from: startDate)
let dateEndString = formatter.string(from: endDate)
let query = "INSERT INTO health_db9 (uuid, sampleType,
sampleValue, startDate, endDate) VALUES (?, ?, ?, ?, ?);"
var statement : OpaquePointer? = nil
if sqlite3_prepare_v2(db, query, -1,
&statement, nil) == SQLITE_OK{
    sqlite3_bind_text(statement, 1,
    (uuid as NSString).utf8String, -1, nil)
    sqlite3_bind_text(statement, 2,
    (sampleType as NSString).utf8String, -1, nil)
    sqlite3_bind_double(statement, 3, sampleValue)
    sqlite3_bind_text(statement, 4,
    (dateStartString as NSString).utf8String, -1, nil)
    sqlite3_bind_text(statement, 5,
```

```
    (dateEndString as NSString).utf8String, -1, nil)
    if sqlite3_step(statement) == SQLITE_DONE {
        print("Data inserted success")
    }else {
        print("Data is not inserted in table")
    }
} else {
  print("Query is not as per requirement")
}
}
```

Táto metóda slúži na pridanie jedného riadku do už vytvorenej tabuľky. Ako vstupné parametre sú premenné, ktoré sa zhodujú s tým, ako je vytvorená štruktúra *healthModelSample*. Obidva dátumové objekty sformátujeme na nami definovaný formát a ten ako *string* pridáme do tabuľky spolu s ostatnými premennými pomocou jednoduchého SQL dopytu.

2.3.3 read

Zdrojový kód 2.12: read

```
func read() -> [healthModelSample]{
var mainList = [healthModelSample]()
let query = "SELECT * FROM health_db9;"
var statement : OpaquePointer? = nil
if sqlite3_prepare_v2(db, query, -1, &statement, nil)
== SQLITE_OK{
    while sqlite3_step(statement) == SQLITE_ROW {
        let uuid = String(describing:
        String(cString:sqlite3_column_text(statement,0)))
        let sampleType = String(describing:
        String(cString:sqlite3_column_text(statement,1)))
        let sampleValue = Double(
        sqlite3_column_double(statement, 2))
        let startDateDB = String(describing:
        String(cString:sqlite3_column_text(statement,3)))
        let endDateDB = String(describing:
        String(cString:sqlite3_column_text(statement,4)))
        let dateFormatter = DateFormatter()
```

```
            dateFormatter.dateFormat = "dd-MM-yyyy HH:mm:ss"
            dateFormatter.timeZone = TimeZone(
            abbreviation: "UTC")
            let startDateFinal = dateFormatter
            .date(from: startDateDB)
            let endDateFinal = dateFormatter
            .date(from: endDateDB)
            let model = healthModelSample(
            uuid: uuid, sampleType: sampleType,
            sampleValue: Double(sampleValue),
            startDate: startDateFinal!,
            endDate: endDateFinal!)
            mainList.append(model)
        }
    }
    return mainList
    }
```

Táto metóda slúži na získanie všetkých záznamov z existujúcej tabuľky. Pomocou jednoduchého SQL dopytu si vyžiadame všetky riadky z tabuľky. Prejdeme po jednom cez každý riadok vo *while* cykle. V tomto cykle definujeme, ako sa má uložiť hodnota z konkrétneho riadku a konkrétneho stĺpca do premennej v programe. Tento proces je priamočiary až na prípad, kde musíme zo *stringu* premeniť tento údaj na dátumový objekt, ale to dokážeme pomocou *formatteru*. Pomocou týchto premenných potom vytvoríme *model*, čo je vlastne premenná štruktúrovaného typu *healthModelSample*. Takto vytvorený *model* pridáme do poľa a na konci metódy naplnené pole vrátime.

2.4 ViewController

Zdrojový kód 2.13: ViewController

```
class ViewController: UIViewController{
@IBOutlet weak var addButton: UIButton!
@IBOutlet weak var showButton: UIButton!
@IBOutlet weak var synButton: UIButton!
override func viewDidLoad() {
    super.viewDidLoad()
```

```
HealthStore.shared.requestAuthorization{success in
    if success{
        print("Authorization successful")
    }else{
        print("Authorization unsuccessful")
    }
}
view.backgroundColor =
hexStringToUIColor(hex: "#0D051B")
addButton.setTitle("Add Value", for: .normal)
addButton.setTitleColor(.black, for: .normal)
addButton.backgroundColor =
hexStringToUIColor(hex: "#4FD3C4")
addButton.layer.cornerRadius = 25
addButton.titleLabel?.font =
UIFont(name: "GillSans", size: 40)
addButton.titleLabel?.minimumScaleFactor = 0.2
addButton.titleLabel?.numberOfLines = 1
addButton.titleLabel?.adjustsFontSizeToFitWidth = true
addButton.titleLabel?.textAlignment = .center
showButton.setTitle("Show Value", for: .normal)
showButton.setTitleColor(.black, for: .normal)
showButton.backgroundColor =
hexStringToUIColor(hex: "#4FD3C4")
showButton.layer.cornerRadius = 25
showButton.titleLabel?.font =
UIFont(name: "GillSans", size: 40)
showButton.titleLabel?.minimumScaleFactor = 0.2
showButton.titleLabel?.numberOfLines = 1
showButton.titleLabel?.adjustsFontSizeToFitWidth
= true
showButton.titleLabel?.textAlignment = .center
synButton.setTitle("Synchronize", for: .normal)
synButton.setTitleColor(.black, for: .normal)
synButton.backgroundColor =
hexStringToUIColor(hex: "#4FD3C4")
synButton.layer.cornerRadius = 25
```

```
        synButton.titleLabel?.font =
        UIFont(name: "GillSans", size: 40)
        synButton.titleLabel?.minimumScaleFactor = 0.2
        synButton.titleLabel?.numberOfLines = 2
        synButton.titleLabel?.adjustsFontSizeToFitWidth = true
        synButton.titleLabel?.textAlignment = .center
    }
    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        navigationController?.setNavigationBarHidden(
        true, animated: animated)
    }
    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)
        navigationController?.setNavigationBarHidden(
        false, animated: animated)
    }
}
```

Tento kontrolér sa stará o pohľad, ktorý je prvou obrazovkou čo užívateľ vidí po otvorení aplikácie. Hneď pri jeho načítaní sa volá metóda ohľadom autorizácie prístupu k zdravotným dátam. Ďalej tu sú tri tlačidlá, ktoré sú programovo definované ako graficky majú vyzerať. V metódach *viewWillAppear* a *viewWillDisappear* sa rieši len to, či má navigačný panel byť viditeľný alebo nie. Navigačný panel nepotrebujeme v tomto pohľade, ale v ďalších áno, tak preto pri príchode na tento pohľad sa tento panel schová, a keď sa z pohľadu odíde, tak sa objaví.

2.5 AddController

Tento kontrolér zodpovedá za pohľad, v ktorom užívateľ pridáva nové zdravotné dáta podľa potreby.

2.5.1 pickerView rozšírenie

Zdrojový kód 2.14: pickerView rozšírenie

```
extension AddController: UIPickerViewDelegate,
UIPickerViewDataSource {
func numberOfComponents(in pickerView: UIPickerView)
```

```
-> Int {
    return 1
}
func pickerView(_ pickerView: UIPickerView,
numberOfRowsInComponent component: Int)
-> Int {
    return healthTypes.count
}
func pickerView(_ pickerView: UIPickerView,
titleForRow row: Int, forComponent
component: Int) -> String? {
    return healthTypes[row]
}
func pickerView(_ pickerView: UIPickerView,
didSelectRow row: Int, inComponent
component: Int) {
    selectedType = healthTypes[row] as String
    print(selectedType)
}
}
```

Tu vidíme niekoľko *delegate* metód, ktoré slúžia na prácu s *pickerView*, ktorý je v hornej časti obrazovky. Ide tu vlastne len o to, aké dáta sú zobrazené v *pickerView*. Jednou z hlavných častí je posledná metóda, ktorá slúži na to, aby sa do premennej *selectedType* zapísala aktuálne vybraná hodnota v rámci tohto *pickerView*. Podobné metódy s inými dátami sú implementované aj v ďalších kontroléroch a to konkrétne *ShowDataView* a *SynDataView*.

2.5.2 textField rozšírenie

Zdrojový kód 2.15: textField rozšírenie

```
extension AddController: UITextFieldDelegate{
func textField(_ textField: UITextField,
shouldChangeCharactersIn range: NSRange,
replacementString string: String) -> Bool {
    let isNumber = CharacterSet.decimalDigits
    .isSuperset(of: CharacterSet(charactersIn: string))
    let withDecimal = (
```

```
            string == NumberFormatter().decimalSeparator &&
            textField.text?.contains(string) == false
        )
        return isNumber || withDecimal
}
}
```

Rozšírenie *textField* je podobne ako metóda vyššie tiež *delegate* metódou a slúži na to, aby pri zadávaní hodnoty do poľa *textField* užívateľ mohol zadať len platný vstup. Pod platným vstupom sa rozumie celé číslo alebo číslo s jednou desatinnou čiarkou.

2.5.3 AddController premenné

Zdrojový kód 2.16: AddController premenné

```
let healthTypes = ["Water","Steps","Distance Ran",
"Distance Cycled","Distance Swam",
"Distance in wheelchair","Calories"]
@IBOutlet weak var elementPicker: UIPickerView!
@IBOutlet weak var dateStart: UITextField!
@IBOutlet weak var dateStartTime: UITextField!
@IBOutlet weak var dateEnd: UITextField!
@IBOutlet weak var dateEndTime: UITextField!
@IBOutlet weak var inputText: UITextField!
@IBOutlet weak var shownValue: UITextField!
@IBOutlet weak var addButton: UIButton!
let datePickerStart = UIDatePicker()
let datePickerStartTime = UIDatePicker()
let datePickerEnd = UIDatePicker()
let datePickerEndTime = UIDatePicker()
var selectedType = "Water"
@IBOutlet weak var startDateStack: UIStackView!
@IBOutlet weak var endDateStack: UIStackView!
```

Na začiatku kontroléra vytvoríme všetky potrebné premenné, ktoré budeme využívať. Väčšina premenných je prepojená s UI elementom v danom pohľade, za ktorý tento kontrolér zodpovedá. Okrem toho tu určíme pole hodnôt, z ktorých bude na výber v rámci pickerView a východziu premennú, ktorá predstavuje to

aká hodnota je vybraná na začiatku, čo je v tomto prípade *Water*. Ďalšou podstatnou časťou sú *UIDatePicker* elementy, ktoré sa zobrazia po stlačení *textField* ku ktorému prislúchajú. Podobne ako vo *ViewController* tak aj v tomto sú programovo definované grafické prvky týchto elementov.

2.5.4 CreateDatePicker

Zdrojový kód 2.17: CreateDatePicker

```
func createDatePickerStart(){
let toolbarStart = UIToolbar()
toolbarStart.sizeToFit()
toolbarStart.barStyle = .default
toolbarStart.barTintColor = UIColor.black
toolbarStart.isTranslucent = false
let doneBtnStart = UIBarButtonItem(barButtonSystemItem:
.done, target: nil, action: #selector(donePressedStart))
toolbarStart.setItems([doneBtnStart], animated: true)
toolbarStart.isUserInteractionEnabled = true
dateStart.inputAccessoryView = toolbarStart
datePickerStart.preferredDatePickerStyle = .inline
datePickerStart.sizeToFit()
datePickerStart.datePickerMode = .date
dateStart.inputView = datePickerStart
}
```

V každom kontroléry sú štyri takéto funkcie. Táto funkcia slúži na vytvorenie pohľadu *datePicker*. Špecifikujeme, ako má vyzerať horný panel a tlačidlo *Done*, ktoré je v ňom umiestnené. Ďalej určíme, aký štýl a mód pre tento *datePicker* chceme. Mód sa mení podľa toho, či chceme mať na výber dátum alebo čas. V každom kontroléry sú štyri takéto funkcie preto, lebo vždy potrebujeme začiatočný dátum, začiatočný čas, konečný dátum a konečný čas. Okrem vybraného módu sa tieto funkcie nelíšia.

2.5.5 Done tlačidlo

Zdrojový kód 2.18: Done tlačidlo

```
@objc func donePressedStart(){
let formatterStart = DateFormatter()
```

```
formatterStart.dateStyle = .medium
formatterStart.dateFormat = "d MMM yyyy"
dateStart.text = formatterStart.string
(from: datePickerStart.date)
self.view.endEditing(true)
}
```

Tu sa vlastne len určuje, čo sa má stať po stlačení tlačidla *Done*, ktoré je vo vrchnom panely pri každom zobrazení *datePicker*. Dátumový objekt z prislúchajúceho *datePicker* prevedieme pomocou *formatteru* do nami požadovanej *string* podoby a do prislúchajúceho *textField* dáme túto hodnotu, a následne schováme *datePicker* pohľad. V každom kontroléry je toto tlačidlo vytvorené štyrikrát z rovnakého dôvodu ako *CreateDatePicker* funkcia.

2.5.6 addClicked funkcia

Zdrojový kód 2.19: addClicked funkcia

```
@IBAction func addClicked(_ sender: UIButton) {
var currentDate = "15 Dec 1998"
let formatterCurrentDate = DateFormatter()
formatterCurrentDate.dateStyle = .medium
formatterCurrentDate.dateFormat = "d MMM yyyy"
currentDate = formatterCurrentDate.string(from: Date())
var currentTime = "11:00"
let formatterCurrentTime = DateFormatter()
formatterCurrentTime.timeStyle = .short
formatterCurrentTime.dateFormat = "HH:mm"
currentTime = formatterCurrentTime.string(from: Date())
let dateFormatterCombine = DateFormatter()
dateFormatterCombine.dateFormat = "d MMM yyyy 'at' HH:mm"
let startDateTemp = (dateStart.text ?? currentDate) +
" at " + (dateStartTime.text ?? currentTime)
let startDateFinal = dateFormatterCombine.date
(from: startDateTemp)
let dateFormatterCombine2 = DateFormatter()
dateFormatterCombine2.dateFormat = "d MMM yyyy 'at' HH:mm"
let endDateTemp = (dateEnd.text ?? currentDate) +
" at " + (dateEndTime.text ?? currentTime)
```

```
let endDateFinal = dateFormatterCombine.date
(from: endDateTemp)
var popUpWindow: PopUpWindow!
switch selectedType {
case healthTypes[0]:
    HealthStore.shared.writeWater(waterConsumed:
    (inputText.text! as NSString).doubleValue, dateStart:
    startDateFinal1!, dateEnd: endDateFinal1!){success in
        if success{
            DispatchQueue.main.async {
                popUpWindow = PopUpWindow(title:"Success",
                text: self.inputText.text! + " ml of water
                successfully added.", buttontext: "OK")
                self.present(popUpWindow, animated: true,
                completion: nil)
            }
            print("Water saved successfully")
        }else{
            DispatchQueue.main.async {
                popUpWindow = PopUpWindow(title:
                "Error", text: "Water was not added.",
                buttontext: "OK")
                self.present(popUpWindow,
                animated: true, completion: nil)
            }
            print("Water was not saved successfully")
        }
    }
}
```

Táto funkcia sa volá po stlačení tlačidla na pridanie hodnoty. Keďže v našom riešení máme jeden dátum rozdelený na dve časti a to dátum a konkrétny čas, tak na začiatku funkcie tieto dva k sebe prislúchajúce dátumové objekty musíme spojiť do jedného, a to s pomocou *formatteru* a tieto vytvorené dátumové objekty sú použité ako vstupné parametre do volanej funkcie nižšie. O to, aká konkrétna hodnota sa má zapísať, a ktorá metóda sa má zavolať sa stará *switch*. V tomto prípade tu je len jeden *case* na ukážku, ale reálne tam je týchto *case* blokov viacero, a každý *case* blok vyzerá rovnako, len s tým rozdielom, že sa zavolá iná metóda

z triedy HealthStore, ktoré som popisoval vyššie. *Case* sa vykoná podľa toho, aká hodnota je vybraná v rámci *pickerView* v hornej časti obrazovky. Po pridaní hodnoty sa následne zobrazí *popUp* okno s informáciou o tom, aký typ, a aká hodnota bola pridaná, a či proces prebehol úspešne alebo nie.

2.6 ShowController

Tento kontrolér je zodpovedný za pohľad, v ktorom si užívateľ dokáže zobraziť vyžiadané dáta. Veľká časť kódu v tomto kontroléry je takmer identická s kódom v *AddController*, a to hlavne to, akým spôsobom pracujeme s dátumový objektami a celkovo ako vytvárame pohľady pre dátum. Funkcia, ktorá sa volá po stlačení tlačidla na zobrazenie je tiež skoro identická a obsahuje podobný *switch* blok, v ktorom jediný rozdiel je to, aká metóda sa zavolá z HealthStore. Tieto metódy sa líšia v tom, že vracajú nejakú hodnotu a *ShowController* túto hodnotu zapíše do *textField*. Ďalším rozdielom je to, že v tomto kontroléry nie sú žiadne *popUp* okná, keďže užívateľ po stlačení tlačidla vidí vyžiadanú hodnotu v *textField* a kvôli tomu nie je potrebné užívateľovi oznámiť, či všetko prebehlo úspešne alebo nie. *Delegate* funkcia pre *textField* tu implementovaná nie je, keďže jediné, čo užívateľ zadá ako vstup je začiatočný dátum, konečný dátum a typ hodnoty, akú chce zobraziť a kvôli tomu nemusíme kontrolovať platnosť žiadneho zo vstupov.

2.7 SynController

Tento kontrolér je zodpovedný za pohľad, v ktorom užívateľ môže synchronizovať vybrané zdravotné dáta z vybraného časového obdobia s lokálnou databázou.

2.7.1 childView

Zdrojový kód 2.20: childView

```
let childView = UIHostingController(
rootView: TaskEditView())
addChild(childView)
childView.view.frame = theContainer.bounds
theContainer.addSubview(childView.view)
```

V rámci *viewDidLoad* funkcie pre tento kontrolér takýmto spôsobom pridáme SwiftUI riešenie do našej aplikácie, ktorá okrem tohto prípadu všade využíva UIKit. UIKit neponúka riešenie pre výber viacerých hodnôt z jedného poľa, a

práve kvôli tomu sme pridali riešenie pre výber viacerých hodnôt z poľa s pomocou SwiftUI. Jedná sa tu vlastne len o pridanie pohľadu, ktorý nemusí byť nutne spravený v SwiftUI do kontajneru, ktorý existuje v *Storyboarde*.

2.7.2 UserDefaults

Zdrojový kód 2.21: UserDefaults

```
let userDefault = UserDefaults.standard
databaseSelected = ((userDefault.value(
forKey: "databaseSelected") as? String ?? "Database1"))
selectedRow = (userDefault.value(
forKey: "selectedRow") as? Int ?? 0)
UserDefaults.standard.synchronize()
```

Tento kód slúži na uloženie vybranej databázy do predvolených hodnôt zariadenia. To znamená to, že ak vyberieme nejakú databázu a vypneme aplikáciu, tak pri jej opätovnom otvorení bude vybraná tá databáza, ktorá bola vybraná naposledy.

2.7.3 synchronize

Zdrojový kód 2.22: synchronize

```
let group = DispatchGroup()
var insertOrNot = true
if(SynData.shared.selectedGoals
.contains(Goal(name: "Water"))){
group.enter()
HealthStore.shared.readWaterForDB(dateStart:
startDateFinal!, dateEnd: endDateFinal!){success in
    if success{
        DispatchQueue.main.async {
            self.db.listOfHealthData.removeAll()
            self.db.listOfHealthData = self.db.read()
            for healthSample in HealthStore.shared
            .samplesToSyncWater{
                print(healthSample.sampleValue)
                insertOrNot = true
                for healthEntry in
```

```
                self.db.listOfHealthData{
                    if(healthSample.uuid
                    == healthEntry.uuid){
                        insertOrNot = false
                    }
                }
                if(insertOrNot){
                    self.db.insert(
                    uuid: healthSample.uuid,
                    sampleType: healthSample.sampleType,
                    sampleValue: healthSample.sampleValue,
                    startDate: healthSample.startDate,
                    endDate: healthSample.endDate)
                }
            }
            print("Synchronized Water")
            group.leave()
        }
    }else{
        DispatchQueue.main.async {
            print("Big Sync Error Water")
            group.leave()
        }
    }
}
}
group.notify(queue: .main) {
HealthStore.shared.samplesToSyncSteps.removeAll()
HealthStore.shared.samplesToSyncWater.removeAll()
HealthStore.shared
.samplesToSyncDistanceRunning.removeAll()
HealthStore.shared
.samplesToSyncDistanceSwimming.removeAll()
HealthStore.shared
.samplesToSyncDistanceWheelchair.removeAll()
HealthStore.shared
.samplesToSyncDistanceCycling.removeAll()
```

```
HealthStore.shared.samplesToSyncHeartRate.removeAll()
HealthStore.shared.samplesToSyncCalories.removeAll()
self.db.listOfHealthData.removeAll()
var popUpWindow: PopUpWindow!
var stringSuccessSyn = ""
if SynData.shared.selectedGoals.count
== 1 && SynData.shared.selectedGoals
.first!.name != "None"{
    print(SynData.shared.selectedGoals.first!.name)
    stringSuccessSyn = SynData.shared
    .selectedGoals.first!.name
    popUpWindow = PopUpWindow(title:
    "Success", text: stringSuccessSyn
    + " were successfully synchronized with "
    + self.databaseSelected
    + ".", buttontext: "OK")
    self.present(popUpWindow,
    animated: true, completion: nil)
}else if SynData.shared.selectedGoals.count
== 1 && SynData.shared.selectedGoals.first!.name
== "None"{
    popUpWindow = PopUpWindow(title: "Error", text:
    "There were no elements chosen to be synchronized.",
    buttontext: "OK")
    self.present(popUpWindow,
    animated: true, completion: nil)
}else{
    var isFirst = true
    for goal in SynData.shared.selectedGoals{
        if isFirst{
            stringSuccessSyn = goal.name
            isFirst = false
            continue
        }
        stringSuccessSyn = stringSuccessSyn
        + ", " + goal.name
    }
```

```
    popUpWindow = PopUpWindow(title:
    "Success", text: stringSuccessSyn
    + " were successfully synchronized with "
    + self.databaseSelected
    + ".", buttontext: "OK")
    self.present(popUpWindow,
    animated: true, completion: nil)
}
}
```

Funkcia *synchronize* sa volá po stlačení tlačidla na synchronizovanie. Keďže synchronizácia sa vykonáva asynchrónne pre každú hodnotu a my potrebujeme vedieť, kedy presne synchronizácia skončí, aby sme mohli užívateľovi zobraziť *popUp* okno so správou o úspechu alebo neúspechu, tak využívame *DispatchGroup*() spolu s *.leave*(). To funguje tak, že pomocou funkcie *group.notify*(*queue:.main*) budeme vedieť, kedy sa došlo ku všetkým *.leave*() riadkom kódu, ktoré sú umiestnené vždy na konci už spomenutých funkcií synchronizácie pre každú hodnotu. V ukážke kódu 2.22 je len jeden *if* blok pre synchronizáciu prijatých tekutín, ale takýchto *if* je viacero za sebou pre každú hodnotu, ktorú máme ako možnosť synchronizovať. Algoritmus synchronizácie funguje tak, že sa načítajú do poľa všetky hodnoty na synchronizovanie a do ďalšieho poľa sa načítajú všetky záznamy uložené v databáze. Pomocou dvoch *for* cyklov porovnáme každý záznam s každým, a ak sa daný záznam v databáze nenachádza, tak ho na konci danej iterácie do databázy pridáme. Po synchronizovaný všetkých žiadaných hodnôt sa užívateľovi zobrazí správa o tom, ktoré hodnoty boli úspešne synchronizované, a ak nevybral žiadne, tak ho na to aplikácia upozorní. Ešte pred zobrazením tejto správy sa ale všetky naplnené polia vymažú, aby neostávali zbytočne v pamäti, keď už nie sú potrebné.

2.8 MultiSelector

Zdrojový kód 2.23: Pole pre synchronizáciu

```
let allGoals: [Goal] =
[Goal(name: "Water"),
Goal(name: "Steps"),
Goal(name: "Heart Rate"),
Goal(name: "Distance Running"),
```

```
Goal(name: "Distance Swimming"),
Goal(name: "Distance in wheelchair"),
Goal(name: "Distance Cycling"),
Goal(name: "Calories")]
```

Takto vytvoríme pole, do ktorého dáme hodnoty, ktoré je možné synchronizovať v rámci aplikácie.

Zdrojový kód 2.24: Vytvorenie tlačidla

```
ForEach(options) { selectable in
    Button(action: { toggleSelection(selectable:
    selectable) }) {
        HStack {
            Text(optionToString(selectable))
            .foregroundColor(.accentColor)
            Spacer()
            if selected.contains
            { $0.id == selectable.id } {
                Image(systemName: "checkmark")
                .foregroundColor(.accentColor)
            }
        }
    }.tag(selectable.id)

}
```

V ukážke kódu 2.24 vytvoríme pre každý prvok vo vyššie spomenutom poli tlačidlo, ktoré je možné stlačiť a umiestnime ich do *HStack*.

2.8.1 toggleSelection

Zdrojový kód 2.25: toggleSelection

```
private func toggleSelection(selectable: Selectable) {
    if let existingIndex =
    selected.firstIndex(where:
    { $0.id == selectable.id }) {
        selected.remove(at: existingIndex)
        if selected.isEmpty == true{
            selected.insert(
```

```
                SynData.shared.goalNone[0] as! Selectable)
            }
        } else {
            selected.insert(selectable)
            if let indexOfNone = selected
            .firstIndex(where: { $0.id as! String
            == SynData.shared.goalNone[0].id }) {
                selected.remove(at: indexOfNone)
            }
        }
        SynData.shared.selectedGoals = selected as! Set<Goal>
        print(SynData.shared.selectedGoals)
}
```

Táto funkcia sa volá po každom stlačení jedného z vyššie spomínaných tlačidiel, ktoré sú na výber pre synchronizáciu. Ak je daný element už vybraný, tak sa odstráni zo setu a naopak, ak nie je vybraný, tak sa do setu pridá. Tento set, ako je už vyššie spomenuté je uložený v *SynData* triede.

2.9 PopUpWindow

Zdrojový kód 2.26: PopUpWindow

```
private let popUpWindowView = PopUpWindowView()
init(title: String, text: String, buttontext: String) {
    super.init(nibName: nil, bundle: nil)
    modalTransitionStyle = .crossDissolve
    modalPresentationStyle = .overFullScreen
    popUpWindowView.popupTitle.text = title
    popUpWindowView.popupText.text = text
    popUpWindowView.popupButton.setTitle(
    buttontext, for: .normal)
    popUpWindowView.popupButton.addTarget(
    self, action: #selector(dismissView),
    for: .touchUpInside)
    view = popUpWindowView
}
```

Takto vytvárame pohľad pre *popUp* okno. Na to, aby sme ho mohli použiť viackrát v rôznych častiach aplikácie sme dali ako vstupné premenné tri textové prvky a to *title*, *text* a *buttontext*. Tieto textové prvky môžu byť čokoľvek, ale my to využívame na to, aby užívateľ vedel, či daná funkcia prebehla v poriadku, potom zadáme samotný obsah správy a ako text tlačidla dáme len potvrdzujúcu správu ako *OK*. Po kliknutí na dané tlačidlo *popUp* okno zmizne.