VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

TRÉNINKOVÝ MANAŽER PRO IOS

TRAINING MANAGER FOR IOS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ONDREJ KONDEK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MARTIN HRUBÝ, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Kondek Ondrej**
Program: Informační technologie
Název: **Tréninkový manažer pro iOS**
Training Manager for iOS
Kategorie: Uživatelská rozhraní

Zadání:

1. Prostuduje programování aplikací pro iOS a knihovny pro podporu sledování zdravotních a pohybových aktivit.
2. Navrhněte aplikaci cílenou na sběr dat o pohybových aktivitách na zařízení iPhone. Zaměřte se na maximalizaci uživatelského komfortu při zadávání dat (zapojení Siri, widgetů na obrazovce apod). Aplikace bude zjištěná data přehledně zobrazovat a umožní sdílení dat mezi jinými uživateli.
3. Aplikaci implementujte.
4. Testujte aplikaci se zapojením několika uživatelů.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- První dva body.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Hrubý Martin, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 11. listopadu 2020

Abstrakt

Táto bakalárska praća sa zaoberá návrhom a implementáciou aplikácie pre operačný systém iOS. Jedná sa o aplikáciou zameranú na zaznamenávanie fyzických aktivivít používateľa, pričom dôraz je kladený na jednoduché zadávanie dát s využitím rozšírení ako hlasová asistentka Siri, widget či kontextové menu. Aplikácia umožňuje zaznamenávať čas vykonávania športových aktivít, ponúka ich prehľad, ale umožňuje aj zdieľanie týchto dát medzi používateľmi. Práca popisuje existujúce nástroje pre vývoj iOS aplikácií, rozbor konkurenčných riešení, návrh a implementáciu samotnej aplikácie a na záver jej testovanie.

Abstract

This bachelor thesis describes a draft and the implementation of an iOS application. The application is meant to work as a tracking tool of user's physical activities with emphasis on simple data input using extensions such as voice assistent Siri, widget and context menu. The user can create records of their activities, show their own statistics but they are also able to share data between other users of the application. The thesis describes existing tools for iOS development, analysis of competitive applications, draft and implementation of the application and in the end its testing.

Kľúčové slová

iOS aplikácia, Apple, iOS, Mobilná aplikácia, iPhone, Widget, Siri, MVC, CloudKit, iCloud, CoreData, Swift, Charts

Keywords

iOS application, Apple, iOS, Mobile Application, iPhone, Widget, Siri, MVC, CloudKit, iCloud, CoreData, Swift, Charts

Citácia

KONDEK, Ondrej. *Tréninkový manažer pro iOS*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Martin Hrubý, Ph.D.

Tréninkový manažer pro iOS

Prehlásenie

Prehlasujem, že túto bakalársku prácu som vypracoval samostatne pod vedením pána Ing. Martina Hrubého, Ph.D. V práci som uviedol všetky literárne pramene, publikácie a zdroje, z ktorých som pri vypracovávaní čerpal.

.....
Ondrej Kondek
11. mája 2021

Poďakovanie

V prvom rade by som chcel poďakovať svojmu vedúcemu tejto bakalárskej práce Ing. Martinovi Hrubému, Ph.D. za odbornú pomoc, cenné rady tykajúceho sa nielen písania tejto práce, ale aj implementačnej časti aplikácie. Ďalej by som sa chcel poďakovať svojej priateľke Rebeke za pomoc s nakreslením obrázkov použitých v aplikácii a jej postrehy ohľadom mojej práce. Rád by som spomenul ešte brata Jakuba a jeho manželku Sašku, ktorí mi poskytli svoje cenné skúsenosti z písania diplomových prác, ktoré som mohol zakomponovať do tejto práce. Na záver chcem už len vysloviť vďaku svojim rodičom za ich psychickú, materiálnu podporu a trpezlivosť počas celého štúdia.

Obsah

Kapitola 1

Úvod

Pokročilé technológie otvárajú ľudstvu nové cesty a stávajú sa spoločníkmi, ktorí uľahčujú, či automatizujú každodenné aktivity. Medzi ne patria napríklad aj mobilné telefóny, ktoré už niekoľko rokov neslúžia len ako komunikačný prostriedok. Tieto zariadenia využívajúc aplikácie ponúkajú mnoho možností pre zábavu, relax, vzdelávanie či prácu. Práve aplikácie sú prostriedkom, vďaka ktorému je veľmi jednoduché rozširovať funkcionalitu mobilných telefónov.

Táto bakalárska práca sa zaoberá štúdiom technológií používaných pre vývoj iOS aplikácií a súčasne návrhom a implementáciou mobilnej aplikácie pre daný operačný systém. Aplikácia, ktorá je predmetom tejto práce, má funkciu tréningového manažéra, ktorý umožňuje používateľovi zaznamenávať športové aktivity. Motiváciou vytvorenia aplikácie je priniesť na trh produkt, ktorý sa odlíši od konkurenčných riešení tým, že je zameraný na špecifický druh športových aktivít. Jedná sa o športové aktivity, pri ktorých je sledovaný predovšetkým čas respektíve doba ich vykonávania. Aplikácia je navrhnutá s cieľom ponúknuť moderné a intuitívne užívateľské prostredie s dôrazom na komfortné zadávanie dát so zapojením widgetov a hlasovej asistentky Siri. Aplikácia ďalej umožňuje používateľovi jednotlivé záznamy upravovať či mazať. Používateľ vďaka nej môže sledovať štatistiky vlastnej športovej aktivity, či môže zobraziť prehľad aktivity iných používateľov. Kľúčovou vlastnosťou aplikácie je však aj možnosť spätného zadávania dát.

Táto práca sa venuje jednotlivým častiam, ktoré viedli k vytvoreniu danej aplikácie. V kapitole 2 sú dôkladne preskúmané rôzne natívne a externé nástroje a frameworky používané pri vývoji iOS aplikácií, ale aj existujúce riešenia podobné tejto aplikácii. Návrhom aplikácie sa zaoberá kapitola 3 a samotnú implementáciu popisuje kapitola 4. Predposledná kapitola 5 obsahuje stručný popis procesu testovania, ktorý vedie k záveru tejto práce zhrnutom v kapitole 6.

Kapitola 2

Prieskum nástrojov a trhu iOS aplikácií

Pre vývoj iOS aplikácií je k dispozícii široký výber rôznych frameworkov, knižníc, vývojových prostredí a architektúr. Výber týchto nástrojov je nutné vykonávať s ohľadom na potreby konkrétnej aplikácie. Vhodne zvolené nástroje umožňujú efektívnu implementáciu ale aj testovanie danej aplikácie.

V nasledujúcej kapitole sú najprv opísané technológie použité pri implementácii (4) aplikácie Tajmer (3.1) a v druhej časti jej konkurečné existujúce riešenia.

2.1 Programovacie jazyky

Pri vývoji natívnych iOS aplikácii sú využívané dva programovacie jazyky - Swift a Objective-C.

2.1.1 Jazyk Objective-C

Objective-C je objektovo orientovaný programovací jazyk, ktorý je rozšírením jazyka C - obohacuje ho o systém zasielania správ jazyku Smalltalk. Tento jazyk bol pôvodne vyvinutý v osemdesiatych rokoch minulého storočia a bol základným jazykom podporovaným Apple - macOS a iOS. Neskôr bol však nahradený práve jazykom Swift.

2.1.2 Jazyk Swift

Ako uvádza oficiálna dokumentácia [1] Swift je širokoúčelový programovací jazyk vyvinutý s dôrazom na moderný prístup k bezpečnosti, výkonu a dizajnovým štruktúram. Tento jazyk bol po prvýkrát predstavený v roku 2014 spoločnosťou Apple Inc. Motiváciou predstavenia nového jazyku bolo nahradiť starší existujúci jazyk Objective-C, ktorý nedisponoval dostatočnými modernými vlastnosťami v porovnaní s inými jazykmi. Swift je založený na príncipoch jeho predchodcu, vďaka čomu je možné používať Objective-C popri jazyku Swift v rámci jedného zdrojového kódu.

Jazyk Swift sa postupom času vyvíjal a dnes je podporovaný nielen Apple platformami (iOS, iPadOS, macOS, tvOS, watchOS), ale aj Windows, Linux a Android - nie je však štandardne využívaný.

2.2 Vývojové prostredie XCode

XCode je grafické vývojové prostredie (IDE) od firmy Apple používané k vytváraniu softvéru pre operačné systémy iOS, iPadOS, macOS, tvOS a watchOS. Prvýkrát bolo predstavené v roku 2003 a dodnes je pravidelne aktualizované (aktuálne najnovšia verzia 12.4.) a voľne dostupné v App Store.

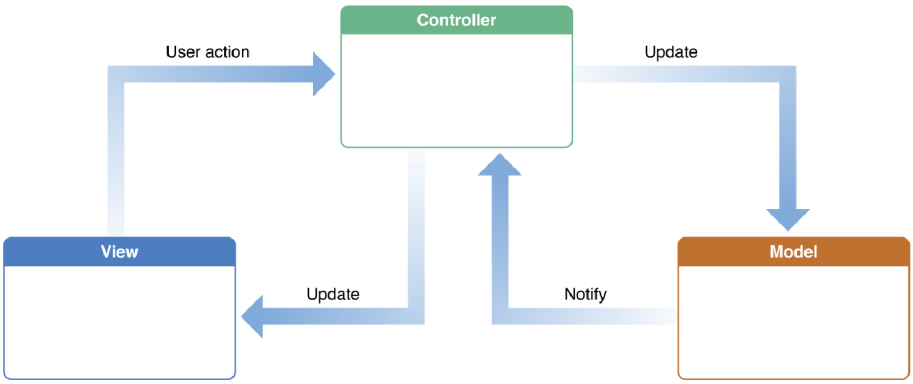
XCode ponúka množstvo nástrojov pre efektívne programovanie a testovanie aplikácií: textový editor, debugger, compiler, simulátor a podobne. Vďaka Xcode je však možné sledovať aj výkon aplikácie, tvoriť grafické rozhranie, testovať na fyzickom zariadení ale aj uverejniť aplikáciu na AppStore. Okrem natívnych jazykov pre tvorbu Apple aplikácií (Swift, Objective-C) podporuje Xcode aj zdrojové kódy jazykov: C, C++, Java, AppleScript, Python, Ruby a ResEdit.

2.3 Vývojový vzor Model-View-Controller

Model-View-Controller (ďalej MVC) je jeden z architektonických vzorov používaných pri vývoji mobilných i počítačových aplikácií.
MVC sa skladá z 3 vrstiev:

- Model - zapúzdruje dáta do aplikácie a definuje logiku a operácie nad nimi
- View - je vizuálna reprezentácia dát, ktorá dokáže spracovávať určité podnety používateľa - upravovanie dát (napr. graf, tabuľka, label a pod.)
- Controller - pracuje ako prostredník medzi aplikačnými view objektami a model objektami. Jeho úlohou je reagovať na podnety a teda aktualizovať stav modelu v prípade zmeny view a naopak.

Ako popisuje Apple vo svojej dokumentácii o MVC [7], vzor nedefinuje len úlohy týchto objektov, definuje spôsob komunikácie medzi nimi. Každá z vrstiev je oddelená od ostatných abstraktnou hranicou a komunikuje s ostatnými cez ňu. MVC je vhodnou architektúrou pre vývoj aplikácií, nakoľko umožňuje oddeliť spracovanie, ukladanie, úpravu a zobrazovanie dát do logických celkov. Takéto aplikácie sú potom ľahšie znovupoužiteľné, rozšíriteľné a prehľadnejšie. Nasledujúci Obrázok 2.1 prehľadne zobrazuje architektúru MVC a vzťahy medzi jednotlivými vrstvami.



Obr. 2.1: Architektúra MVC [7]

2.4 Natívne nástroje

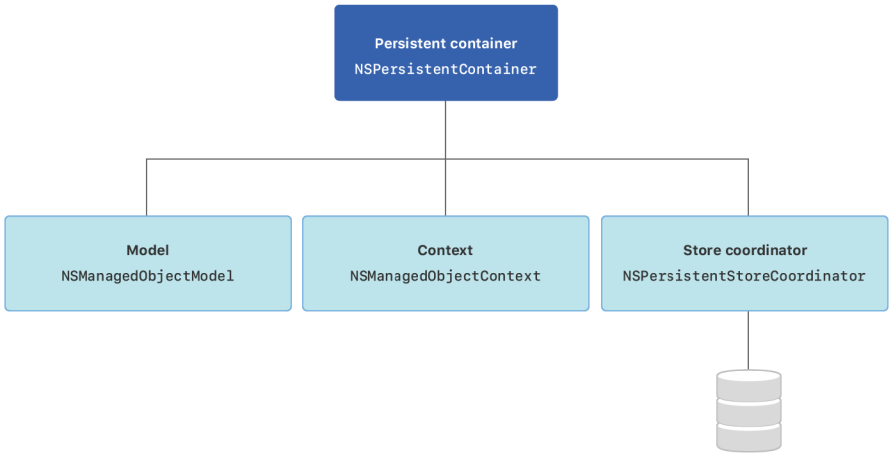
2.4.1 Framework Core Data

Core Data je framework používaný k práci s objektami z modelovej vrstvy aplikácie. Poskytuje automatizované riešenia pre prácu s lokálnou databázou - uloženie, sledovanie, úprava a filtrovanie dát. Apple v dokumentácii Core Data [5] uvádza, že programátor vďaka tomuto frameworku typicky ušetrí 50 až 70 percent množstva kódu, ktorý by inak musel písať manuálne. O perzistenciu v Core Data sa v pozadí stará SQLite3. Perzistentý objekt je taký, ktorého obsah je zapamätaný aj mimo behu aplikácie, a tak je možné k nemu pri znovuotvorení pristúpiť.

Správa perzistentných objektov je umožnená pomocou objektového kontajnera `ManagedObject`, ktorý je možné chápať ako abstrakciu nad relačnou databázou a súčasne je hlavnou nadtriedou všetkých objektov. Pomocou `NSPredicate` je ďalej možné vytvárať komplexné databázové dopyty na uložené dáta, čo nahradzuje dopytovanie priamo pomocou SQL. Fungovanie aplikácie využívanej Core Data je podmienené existenciou takzvaného Core Data zásobníku. Architektúra Core Data zásobníku je založená na použití 3 základných objektov:

- `NSManagedObjectModel` - objekt držiaci metadáta o databázovej schéme
- `NSManagedObjectContext` - správa všetkých databázových objektov a operácií
- `NSPersistentStoreCoordinator` - spája ostatné komponenty stacku a jeho úlohou je riadenie prístupov na databázové úložisko

Z hľadiska jednoduchšej implementácie je možné využiť `NSPersistentContainer`, ktorý zapúzdruje vytvorenie a správu zásobníku. Štruktúru Core Data zásobníku a vzťahy jeho komponent vyjadruje Obrázok 2.2.



Obr. 2.2: Štruktúra Core Data zásobníku a jeho zapúzdrenie do Persistent kontaineru [3]

2.4.2 Trieda UserDefaults

Pre ukladanie jednoduchých a malých objemov dát - napr. nastavenia používateľa - sa používa trieda `UserDefaults`. `UserDefaults` umožňuje ukladať perzistentné hodnoty vo forme párov kľúč:hodnota.

Apple dokumentácia [4] špecifikuje dátové typy hodnôt, ktoré `UserDefaults` podporuje. Konkrétne sa jedná o štandardné dátové typy jazyku Swift ako: `float, double, integer, boolean, URL` alebo objekt, ktorý musí byť inštancia jednej z tried: `NSData, NSString, NSNumber, NSDate, NSArray alebo NSDictionary`.

2.4.3 Framework CloudKit

CloudKit je framework umožňujúci presun dát medzi aplikáciou a iCloud kontajnermi. Jeho úlohou je predovšetkým uloženie dát na cloud, tak aby bolo možné k nim pristupovať z viacerých zariadení. CloudKit podporuje takéto zdieľanie súkromne (viditeľné dáta len medzi používateľovými zariadeniami), verejne (prístup k dátam majú všetci používatelia), ale taktiež dáva možnosť „shared“ (explicitná špecifikácia prístupov k dátam).

Podľa Apple dokumentácie [2] však tento framework nie je nástrojom, ktorý by nahradzoval funkcionalitu Core Data (2.4.1) či iného frameworku na prácu s databázou. CloudKit slúži výlučne k presúvaniu dát na a z iCloud serverov. Nakoľko poskytuje len minimálnu podporu offline cacheovania, je závislý od internetového pripojenia a súčasne platného iCloud konta. Platné iCloud konto je však nutné len v prípade zápisu dát na iCloud. V prípade čítania dát z cloudu je možné pristupovať k dátam neobmedzene - teda bez účtu iCloud.

Základnou jednotkou CloudKit-u sú záznamy. Záznam je typ `dictionary` kľúč:hodnota dvojíc, ktoré reprezentujú dáta ukladajúce sa na server. Trieda `CKRecord` definuje rozhranie pre prácu so záznamami. Pre správu asynchrónnych požiadavok je využívaná trieda `Operation`.

Základné triedy pre použitie CloudKit-u:

- `CKContainer` - abstrakcia nad databázovým kontainerom cloudu, spracovanie prístupov k obsahu dát na cloude
- `CKDatabase` - abstrakcia konkrétnej databázy kontaineru (private, public, shared)
- `CKRecordZone` - možnosť rozdelovať záznamy do zón - zložky v databáze
- `CKRecord` - záznam v databáze (kľúč:hodnota)
- `NSOperation` - práca so záznamami (uloženie, dotazy, metadátové operácie)

2.4.4 Framework UIKit

UIKit je Apple natívny framework pre operačné systémy iOS a tvOS poskytujúci základné objekty pre implementáciu grafického rozhrania, respektíve grafických elementov nachádzajúcich sa v takzvaných views (obrazovkách) aplikácie. Funkciou týchto elementov je zobrazenie obsahu aplikácie na displej, umožnenie interakcií s obsahom a ich spracovanie. Okrem grafických prvkov poskytuje UIKit aplikáciam podporu giest, animácií, kreslenia, tlačenia, informácie o stave zariadenia, ale aj podporu rôznych rozšírení.

Druhou dôležitou úlohou UIKit je koordinácia behu aplikácie - životný cyklus. O túto funkciu sa stará trieda `UIApplication`, ktorá implementuje metódy protokolu

`UIApplicationDelegate`. Pomocou tohto protokolu je `UIApplication` informovaná o dôležitých udalostiach, ktoré môžu nastať pri behu aplikácie - napr. spustenie, vypnutie, pamäťové upozornenia a podobne.

Medzi základné komponenty, ktoré UIKit obsahuje, patria napríklad:

- UIView
- UILabel
- UIButton
- UIViewController
- UITabBarController
- UINavigationController

2.4.5 SiriKit

SiriKit reprezentuje skupinu nástrojov, ktorá umožňuje iOS a watchOS aplikáciám komunikovať so Siri. Siri je virtuálna hlasová asistentka, ktorá je schopná odpovedať na používateľove podnety. Jej úlohou je pomáhať uskutočniť určité príkazy výlučne hlasom, dokonca aj ak je zariadenie zamknuté.

SiriKit zahŕňa frameworky `Intents` a `IntentsUI`, pomocou ktorých je možné implementovať rozšírenia, ktoré konfigurujú príkazy - respektíve podnety, na ktoré Siri dokáže reagovať. Definuje sa teda podnet, na ktorý Siri reaguje a jej reakcia. Tieto frameworky pomocou rozšírenia Intents App Extension umožňujú navyše sofistikovanejšie príkazy, kedy je Siri schopná požiadať o podrobnejšie informácie alebo dávať viacero možností reakcie na jeden typ podnetu. Pomocou tohto rozšírenia je tak možné so Siri komunikovať obojstranne - napríklad požiadať Siri o vykonanie platby, pričom sa Siri opýta na čiastku, príjemcu a podobne. Bez tohto rozšírenia je komunikácia typu: príkaz - reakcia - napríklad spustiť časovač (hodnota časovača musí byť manuálne zvolená). Dôležitým rozdielom je taktiež to, že vďaka Intents App Extension nie je nutné aplikáciu spustiť a Siri komunikuje nezávisle od samotnej aplikácie.

V rámci obmedzeného Apple developer študentského účtu nie je podporované použitie Intents App Extension, teda aplikácia Tajmer neobsahuje toto rozšírenie.

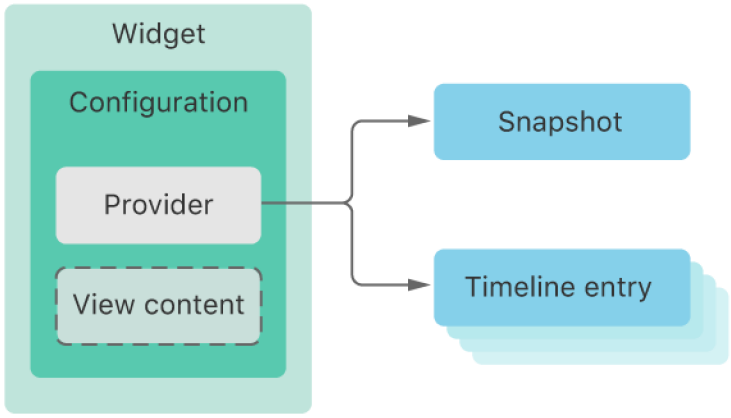
2.4.6 WidgetKit

Apple v júni 2020 predstavil nový operačný systém iOS 14 a spolu s ním takzvané widgety. Widgety nahrádzajú a zároveň rozširujú funckiu dovtedy známych Today Extensions. Jedná sa o aplikačné panely, ktoré je možné umiestniť na Today View (známe aj ako Notification Center) alebo na Home Screen. Ich úlohou je zobrazenie informácií, ktoré aplikácia obsahuje, aj ak sa aplikácia nenachádza aktuálne na popredí. Widget môže obsahovať rôzne grafické elementy - nie však štandardne interaktívne prvky ako napríklad tlačidlo.

WidgetKit je framework umožňujúci widgety implementovať. Pre integráciu widgetu je nutné do aplikácie pridať Widget Extension, následne implementovať views pre zobrazenie obsahu widgetu. Implementácia views musí byť výhradne pomocou frameworku SwiftUI. Posledným krokom je konfigurácia widgetu pomocou timeline provider. Použitím triedy `Timeline` sa definuje ako často respektíve kedy má byť widget aktualizovaný ale aj dáta, ktoré budú zobrazené. Informáciu o aktualizácii widgetu ďalej zabezpečí protokol

`TimelineProvider`. Táto architektúra je zobrazená na Obrázku 2.3, pričom View content označuje grafické rozhranie a Provider označuje spomínaný protokol `TimelineProvider` starajúci sa o zobrazené dáta na widgete. Snapshot je náhľad, teda neresponzívna, statická podoba widgetu zobrazujúca sa napríklad v knižnici widgetov. Timeline entry predstavuje jednotlivé dáta, ktoré provider zobrazuje do widgetu.

Aplikácia môže obsahovať neobmedzený počet rôznych widgetov. WidgetKit ponúka 3 veľkosti widgetu a navyše pomocou SiriKit Intents umožňuje widgety priamo používateľovi upravovať.



Obr. 2.3: Architektúra widgetu [6]

2.5 Externé nástroje

2.5.1 Framework FS Calendar

Pre tvorbu scény s kalendárom je možné využiť voľne dostupný framework FS Calendar. Tento framework umožňuje jednoduchú implementáciu kalendáru vo Swifte pomocou triedy `FSCalendar`, ktorá dedí od triedy `UIView`. FS Calendar ponúka široké možnosti grafických úprav kalendáru priamo pomocou Interface Builder v rámci prostredia XCode. Taktiež obsahuje metódy, pomocou ktorých je možné spraviť kalendár interaktívny, či získavať určité informácie z aktivity používateľa ako napríklad zvolený dátum - o toto sa starajú protokoly `FSCalendarDelegate` a `FSCalendarDataSource`. Framework FS Calendar poskytuje vhodné riešenie, nakoľko natívny framework UIKit takýto element neobsahuje a jeho implementácia by bola časovo náročná.

Ukážka implementácie kalendáru a využitie protokolov vychádzajúca z dokumentácie FSCalendar [10]:

```
// outlet prepojený so Storyboard
@IBOutlet weak var calendarView: FSCalendar!

// nastavenie delegátov
calendarView.delegate = self
calendarView.dataSource = self
```

```
// implementácia metód delegátu
extension ViewController: FSCalendarDelegate, FSCalendarDataSource
{
    func calendar(_ calendar: FSCalendar, didSelect date: Date,
                  at monthPosition: FSCalendarMonthPosition)
    {
        // date - zvolený, označený deň
        print(date)
    }

    func calendar(_ calendar: FSCalendar,
                  numberOfEventsFor date: Date) -> Int
    {
        // nastavenie udalosti pre daný dátum
        // zobrazenie bodky v kalendári pre daný dátum
        // pri všetkých dátumoch v poli dates sa zobrazí 1 bodka
        if dates.contains(date) {
            return 1
        }
        return 0
    }
    ...
}
```

2.5.2 Framework Charts

Charts je voľne dostupný framework tretej strany, ktorý umožňuje jednoduchú tvorbu rôznych grafov. Charts je framework preimplementovaný pre jazyk Swift a Objective-C z pôvodného MPAndroidChart, ktorého autorom je Philipp Jahoda. Tento framework je veľmi obľúbený a populárny, nakoľko ponúka veľmi široké spektrum možností implementácie. Programátor si je schopný vybrať spomedzi 8 typov grafov, ktoré je možné rôzne upravovať (farby, legendy, a podobne), exportovať či animovať. Podobne ako v prípade FS Calendar, Charts ponúka elementy chýbajúce v knižnici UIKit.

Základným prvkom pri implementácii je trieda určujúca druh grafu - napríklad `BarChartView`, `LineChartView`, `PieChartView` a podobne. Takéto view je nutné najprv umiestniť na obrazovku a nasledujúcim krokom je príprava dát pre daný graf. Príprava dát sa skladá z vytvorenia takzvaného datasetu skladajúceho sa z dátových vstupov (entries). Takto pripravený dataset je možné prideliť grafu, ktorý dané dáta adekvátne vykreslí.

Ukážka implementácie stĺpcového grafu pomocou frameworku Charts:

```
// outlet prepojený so Storyboard
@IBOutlet weak var barChart: BarChartView!

func createBarChart() {

    // vytvorenie príkladových data entries
    var entries = [BarChartDataEntry]()
    for x in 0...6 {
```

```
            entries.append(BarChartDataEntry(x: Double(x), y: Double(x)))
        }

        let set = BarChartDataSet(entries: entries, label: "Demo Chart")
        let data = BarChartData(dataSet: set)

         barChart.data = data

        // možná úprava vzhľadu grafu, jednotlivých os a pod.
        barChart.drawValueAboveBarEnabled = false
        barChart.backgroundColor = .white
        set.colors = ChartColorTemplates.joyful()
        ...
}
```

2.6 Rozbor existujúcich riešení

Existuje mnoho rôznych aplikácií, ktorých cieľom je sledovanie pohybových aktivít používateľa. Aplikácie počítajúce kroky, kilometre, sledujúce tep, tlak či iné životné funkcie. Ich zámerom je predovšetkým ponúknuť prehľad o fyzickej aktivite, no súčasne motivovať používateľov k vykonávaniu športových aktivít. Tieto aplikácie nie sú teda zamerané len na profesionálnych športovcov, ale aj na obyčajných používateľov, ktorí chcú získať pravidelne prehľad o svojej fyzickej aktivite.

Rôzne aplikácie však ponúkajú rôzne prístupy, a tak má používateľ na výber mnoho aspektov, ktoré je schopný sledovať. Z tohto dôvodu sa však často aplikácie javia neprehľadné a uživateľsky neintuitívne. Príliš mnoho možností, ktoré používateľ nepotrebuje, sú tak kontraproduktívnymi a používateľa odrádzajú od používania tejto aplikácie. Tieto skutočnosti je dôležité zvážiť pri návrhu takejto aplikácie a preto je nutné dôkladne preštudovať existujúce riešenia. Ďalším z dôvodov je však fakt, že na trhu nie sú potrebné dve aplikácie s rovnakou funkcionalitou, a tak je nutné ponúknuť používateľom niečo unikátne.

V tejto podkapitole je popísaných niekoľko existujúcich riešení na sledovanie športových aktivít používateľa a súčasne ich porovnanie. V poslednej časti sa nachádza porovnanie existujúcich riešení a aplikácie, ktorá je predmetom tejto bakalárskej práce.

2.6.1 Aplikácie Zdravie, Kondícia a Tréning

Pravdepodobne najväčším konkurentom sú aplikácie Zdravie, Kondícia a Tréning od spoločnosti Apple. Všetky tieto aplikácie sú predinštalované na každom zariadení iPhone a podmienkou ich používania je platný iCloud účet. Ich úlohou je sledovanie rôznych fyzických aktivít, zdravia ale i telesné hodnoty používateľa (tlak, tep, atď...). Na základe týchto dát vyhodnocujú rôzne zdravotné hľadiská a sú schopné používateľa usmerniť, či ho upozorniť. Aplikácia Zdravie tvorí základ, ktorého funkcionalitu rozširujú a dopĺňajú samostatné aplikácie Kondícia a Tréning využívajúce inteligentné hodinky AppleWatch.

Aplikácia Zdravie ponúka niekoľko funkcií relevatných pre porovnanie s aplikáciou popísanou v tejto práci. Zdravie umožňuje ukladať záznamy o športových aktivitách (tréningoch), ktoré obsahujú čas, prípadne počet prejdených kilometrov. Takéto záznamy je však nutné vkladať ručne, nie je možné tento proces zautomatizovať s výnimkou počítania krokov. Používateľ je schopný zobraziť štatistiky vo forme grafu. Nie je však umožnené

žiadnym spôsobom zdielať dáta medzi používateľmi. Nakoľko aplikácia obsahuje veľké spektrum funkcií, pre účel sledovania tréningov nemá dostatočne intuitívne grafické prostredie. Funkcie týkajúce sa zdravia ako napríklad - dýchanie, mobilita, srdce, telesné miery síce rozširujú možnosti, no ak používateľ hľadá aplikáciu pre sledovanie športových aktivít, sú tieto funkcie nadbytočné a robia aplikáciu menej prehľadnou.

Súvisiaca aplikácia Kondícia však výrazne dopĺňa funkcionalitu. Jej nevýhodou je ale závislosť na hodinkách AppleWatch. Pomocou hodiniek a aplikácie je možné automaticky ukladať dáta o fyzickej aktivite obsahujúce rôzne atribúty (čas, dátum, miesto, trasa, kilometre, ...). Nevýhodou v porovaní s aplikáciou Zdravie je neschopnosť spätného zadávania dát. V praxi to znamená, že ak si používateľ zabudne hodinky, svoj tréning už spätne nezaznamená. Grafické rozhranie aplikácie ponúka prehľadné grafy, kalendár s aktivitami ale i možnosť porovnávania dát medzi používateľmi. Dokonca je možné vyzvať priateľa na rôzne osobné súťaže. Aplikácia Zdravie importuje rôzne dáta získané aplikáciou Kondícia, avšak nie naopak. To znamená, že aj keď Zdravie umožňuje spätné - ručné zadanie dát, nie je možné takto pridané dáta synchronizovať s ostatnými.

Aplikácia Tréning je obdoba aplikácie Kondícia pre operačný systém systém watchOS. Vďaka tejto aplikácii je používateľ schopný zvoliť aktuálne zaznamenávanie aktivity, jej pozastavenie, ale i počúvať hudbu počas tréningov a podobne. Tréning funguje ako prostredník medzi zariadeniami AppleWatch a iPhone.

Obr. 2.4: Ukážka aplikácie Kondícia [9]

2.6.2 Aplikácia Strava

Strava je multiplatformová mobilná aplikácia pre sledovanie športových aktivít. Je známa predovšetkým svojou schopnosťou zaznamenávať trasy, ale je schopná sledovať aj čas a prevýšenie. Okrem sledovania športovej aktivity ponúka aj sledovanie telesnej aktivity (tep, tlak) za pomoci rôznych externých zariadení. Aplikácia je zameraná predovšetkým na zber informácií o športoch, ktorých cieľom je pohyb z miesta na miesto (beh, cyklistika, turistika a pod.). Strava ponúka veľmi prepracovaný systém zdieľania dát medzi používateľmi, či priame zdieľanie na sociálne siete. Aplikácia v platenej verzii ponúka i podrobné prehľady a analýzy tréningov.

Strava vyniká oproti konkurencii predovšetkým tým, že je dostupná na Android aj iOS platformách. Tento fakt môže z veľkej časti za jej aktuálnu popularitu. Používateľov účet

a jeho dáta sú tak dostupné zo širšieho výberu zariadení a nie je závislý od špecifického účtu - Strava dáva na výber viacero možností autentifikácie. Jej grafické prostredie pôsobí prepracovanejšie v porovnaní s vyššie spomínanými aplikáciami a veľkou výhodou je integrácia s mapami. Po zaznamenaní tréningu dáva možnosť vygenerovať prehľadnú mapu spolu s kľúčovými štatistikami.

Nevýhodou aplikácie je však absencia akéhokoľvek spätného zádavania dát. Ak si teda používateľ zabudne mobilný telefón, nemá možnosť takúto aktivitu zaznamenať. Druhou nevýhodou je veľmi obmedzený počet športov - kategórií, podľa ktorých je možné dáta agregovať.

Obr. 2.5: Ukážka aplikácie Strava [8]

2.6.3 Porovnanie aplikácií

Základom úspechu každej aplikácie je priniesť na trh niečo nové, zaujímavé, inovatívne a súčasne užívateľsky prístupné. Spomínané existujúce aplikácie (Zdravie, Kondícia, Strava) síce ponúkajú mnoho možností a podporujú dokonca i externé zariadenia, no zároveň používateľovi veľmi výrazne obmedzujú zadávanie či upravovanie dát. Preto vznikol koncept novej aplikácie - Tajmer, ktorá je zameraná predovšetkým na pohodlné zadávanie a správu dát. V tejto podkapitole sú popísané základné rozdiely, ale aj spoločné znaky aplikácie Tajmer a jej konkurenčných riešení.

Hlavným cieľom aplikácie Tajmer v porovnaní s inými je sledovanie dĺžky trvania tréningov - čas. Iné aplikácie zaznamenávajú okrem času aj napr. spálené kalórie, odbúraný tuk alebo prejdenú vzdialenosť. Široká škála možností tak často vedie k neprehľadnému grafickému rozhraniu. Aplikácia Tajmer vďaka svojej špecializácii síce neponúka toľko možností, no poskytuje príjemnejšie prostredie pre používateľov, ktorí hľadajú aplikáciu výlučne na tento účel. Kľúčovým rozdielom je možnosť spätného zadávania záznamov o aktivite a ich úprava bez akýchkoľvek obmedzení, čo v konkurenčných riešeniach väčšinou nie je možné vôbec. Aplikácia ponúka plnohodnotné sledovanie vlastných tréningov a umožňuje taktiež sledovanie dát medzi používateľmi. Výraznou nevýhodou je však absencia podpory externých zariadení, ktoré robia sledovanie presnejším. Podrobné porovnanie aplikácií sa nachádza v nasledujúcej tabuľke 2.1.

Funkcionalita	Zdravie	Kondícia	Strava	Tajmer
Tvorenie záznamov	Áno	Áno	Áno	Áno
Úprava záznamov	-	-	Obmedzené*	Áno
Vymazanie záznamov	Obmedzené**	-	Áno	Áno
Spätné pridávanie záznamov	Áno	-	-	Áno
Zdieľanie dát	-	Áno	Áno	Áno
Externé zariadenia	Áno	Áno	Áno	-
Mapy	-	Áno	Áno	-
Widget	-	Áno	Áno	Áno

* aplikácia dokáže upravovať len určité atribúty záznamu - nie je možné upraviť čas aktivity

** aplikácia dokáže vymazať len manuálne vytvorené záznamy

Tabuľka 2.1: Porovnanie aplikácií

Kapitola 3

Návrh aplikácie

Návrh je dôležitým bodom predchádzajúci implementáciu (4). Korektný návrh dokáže ušetriť mnoho zdrojov investovaných do následného vývoja aplikácie, predísť chybám, či lepšie zacieliť apliká̆ciu pre danú skupinu. Preto by mal byť tvorený s ohľadom na niekoľko faktorov ako napríklad: cieľová skupina, pre ktorú bude aplikácia tvorená, dostupné nástroje pre implementáciu, prehľad konkurenčných riešení a používateľské rozhranie. Dôležitým bodom návrhu je však aj modelová časť aplikácie - teda databáza.

Táto kapitola sa zaoberá návrhom aplikácie - Tajmer. Opisuje kľúčové vlastnosti tejto aplikácie (3.1), cieľovú skupinu (3.2), prípady použitia (3.3), na základe ktorých je navrhnuté grafické rozhranie (3.4) a systém ukladania dát (3.5).

3.1 Funkcionalita aplikácie Tajmer

Jedným z cieľov tejto bakalárskej práce je vytvoriť iOS aplikáciu - Tajmer. Motiváciou je ponúknuť aplikáciu slúžiacu na zaznamenávanie športových aktivít s dôrazom na pohodlné zadávanie vstupných dát. Pre zadávanie dát aplikácia okrem vlastného rozhrania podporuje asistentku Siri, Widget a kontextové menu.

Aplikácia umožňuje ukladanie, upravovanie či mazanie záznamov ale aj ich zdieľanie medzi používateľmi. Aplikácia sa zameriava predovšetkým na športové aktivity, pri ktorých používatelia sledujú čas jej vykonávania, respektíve čas zohráva hlavnú úlohu pri zaznamenávaní tejto aktivity. Na základe zbieraných dát je používateľ schopný zobrazovať rôzne štatistiky o svojich aktivitách vo forme grafov a tabuliek.

3.2 Cieľová skupina

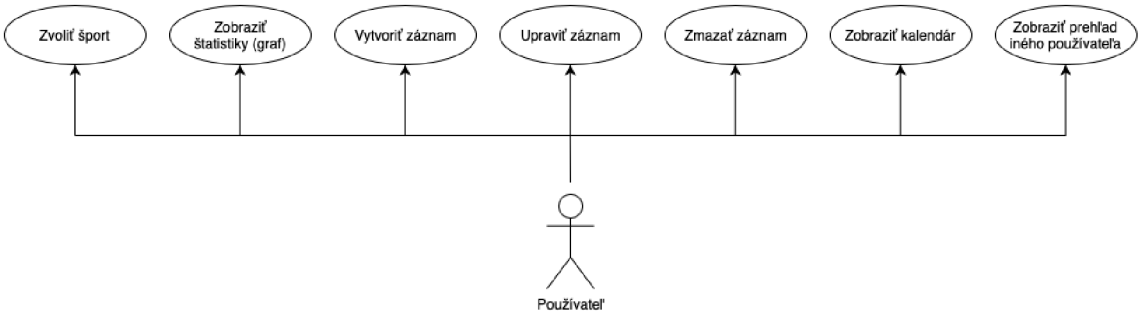
Hlavnou motiváciou pre tvorbu aplikácie je priniesť na trh softvér umožňujúci sledovať športové aktivity používateľov. Napriek množstvu konkurenčných aplikácií neexistuje vhodné riešenie pre používateľov, ktorí hľadajú možnosť zaznamenávať výlučne časové intervaly svojich aktivít, respektíve sledovať čas, ktorý venujú svojim tréningom. Je zrejmé, že bežci, či cyklisti okrem času sledujú aj prejdenú trasu či rýchlosť - teda počet metrov, kilometrov a podobne. Aplikácia popísaná v tejto bakalárskej práci je pre ľudí venujúcim sa športom ako napríklad workout, tanec, biketrial, skateboarding, box a mnoho ďalších športov, pri ktorých je najrelevantnejší faktor čas, ktorý tréningu venujú.

Aplikácia je určená profesionálnym ale aj rekreačným športovcom, ktorí sa snažia udržiavať pravidelnosť svojich športových aktivít. Okrem prehľadu o svojich aktivitách však

získajú prehľad o aktivitách iných používateľov, čo vedie taktiež k motivácii zotrvať pri športe, či potrebu sa zlepšovať. Okrem spomínaných športovcov je však aplikácia vhodná i pre trénerov, ktorí týmto spôsobom môžu získať lepší prehľad, či kontrolu nad svojimi zverencami.

3.3 Prípady použitia

Nasledujúci diagram 3.1 vyjadruje možné interakcie používateľa so softvérom a jeho vzťahy s rôznymi prípadmi použitia aplikácie.



Obr. 3.1: Diagram prípadov použitia

Po spustení aplikácie má používateľ niekoľko možností, ako s ňou pracovať. Základnou možnosťou je výber športu, ktorý určí ostatné interakcie tak, že platia len pre daný šport. To znamená, že ak sa používateľ rozhodne zobraziť štatistiky, bude mu zobrazený prehľad aktuálne vybraného športu. Podobný princíp sa uplatňuje pri ostatných možnostiach. Používateľ má možnosť vytvoriť záznam o aktivite, upraviť alebo taktiež zmazať takýto záznam. Pri tvorbe alebo úprave nastavuje dané atribúty tohto objektu (viac v 3.5.1). Poslednou možnosťou závislou na zvolenom športe je zobrazenie kalendáru. Táto možnosť zobrazí interaktívny kalendár nesúci informácie o športovej aktivite pre každý deň - pomocou tohto kalendáru je ďalej možné záznamy upravovať či mazať. V prípade, že chce používateľ zobraziť všetky športy naraz, systém mu takúto možnosť ponúka priamo v možnosti výberu športu. Tento výber je kedykoľvek možné zmeniť.
Ďalšou možnosťou, na ktorú však voľba športu nemá vplyv, je nájdenie a následné zobrazenie prehľadu záznamov iného používateľa.

3.4 Návrh grafického rozhrania

3.4.1 Aplikácia

Grafické rozhranie aplikácie bolo navrhnuté pomocou voľne dostupného nástroju Figma. Pri návrhu bol braný ohľad na používateľské pohodlie a štandardy dizajnu mobilných iOS aplikácií. Hlavným cieľom návrhu grafického rozhrania je ponúknuť plnohodnotné, moderné a čo najintuitívnejšie prostredie.

Proces návrhu prebiehal v niekoľkých etapách, počas ktorých prebiehalo testovanie reálnymi používateľmi na prototype a na základe spätnej väzby bolo toto grafické rozhranie postupne upravované.

Obr. 3.2: Návrh grafického rozhrania

Návrh na Obrázku 3.2 nie je kompletný a konečný - zobrazuje iba kľúčové prvy grafického rozhrania. Kolorizácia a umiestnenie jednotlivých elementov nekorešponduje presne s finálnym grafickým rozhraním.
Na Obrázku 3.2 je možné vidieť niekoľko obrazoviek - `Views`. Vo vrchnej časti obrázku - prvom riadku sa nachádzajú hlavné `Views`, na ktoré sa je možné dostať pomocou komponenty `TabBar`. `TabBar` je ovládacím panelom tejto aplikácie a nachádza sa v spodnej časti každého `View`. `TabBar` sa nenachádza len na obrazovke, ktorá je prezentovaná takzvane modálne - napríklad obrazovka vľavom dolnom rohu. V tomto prípade je obrazovka prezentovaná cez aktuálnu a prekrýva jej obsah.

Na druhom riadku je možné vidieť `Views`, na ktoré sa nedá presmerovať priamo z komponenty `TabBar`, ale je možné k ním pristúpiť pomocou kliknutia na určité miesto v hlavnom `View`. Tieto spojenia sú na obrázku vyjadrené pomocou červených šípiek.

Popis jednotlivých obrazoviek prvého riadku na Obrázku 3.2 (zľava doprava):

- Zobrazenie štatistík v `ScrollView`
- Zobrazenie kalendáru so športovou aktivitou - konkrétne aktivity pre vybraný deň (v spodnej časti)
- Vytvorenie nového záznamu - možnosť nastavenia času, dátumu, miesta a poznámky
- Vyhľadávanie používateľov
- Nastavenia

Popis obrazoviek druhého riadku na Obrázku 3.2 (zľava doprava):

- Výber aktuálneho športu - možné pristúpiť z ktorejkoľvek hlavnej obrazovky
- Zobrazenie grafu pre vybraný šport - možnosť zobraziť podrobnosti pomocou interaktívneho grafu
- Zobrazenie podrobných informácií o zázname - možnosť upraviť/zmazať/uložiť
- Získanie a výber polohy pre aktuálny záznam
- Zobrazenie štatistík vybraného používateľa na základe výberu z jeho rodičovského `View`

3.4.2 Widget

Okrem aplikácie je nutné navrhnúť grafické rozhranie widgetu, ktorý je dostupný pre používateľa na Home Screen a v Notification Center. Podobne ako pri návrhu grafického rozhrania aplikácie (3.4.1) bol k návrhu použitý nástroj Figma.

iOS umožňuje umiestniť 3 typy (veľkosti) widgetov na obrazovku: 2x2 , 4x2 a 4x4, pričom mierou je počet ikon aplikácií na domovskej obrazovke. Aplikácia Tajmer bude obsahuje jeden widget typu 4x2, ktorého cieľom je ponúknuť používateľovi jednoduchý prehľad o prebiehajúcom tréningu, no zároveň umožňuje tréning spustiť, pozastaviť alebo reštartovať.

Obr. 3.3: Návrh grafického rozhrania widgetu

Obrázok 3.3 prezentuje návrh grafického rozhrania widgetu. V ľavej časti widgetu sa nachádza logo aplikácie. Pravú časť je možné rozdeliť na časť s informačným labelom a

časť obsahujúcu 3 tlačidlá. Informačný panel môže obsahovať 3 rôzne správy - začať nový tréning, tréning spustený v čase X a tréning pozastavený. Prvé tlačidlo (vľavo) umožňuje tréning spustiť/pozastaviť, druhé (v strede) tréning reštartuje a posledné umožňuje vybrať šport.

3.5 Návrh databázy

Databáza aplikácie pozostáva z dvoch častí:

- Lokálna databáza 3.5.1 - databáza uložená priamo v zariadení
- Online databáza 3.5.2 - databáza uložená na cloudovom úložisku iCloud

3.5.1 Návrh lokálnej databázy

Pre správu dát lokálnej databázy je využívaný framework Core Data (2.4.1). Do lokálnej databázy sú vkladané záznamy 2 entít - `Record` a `UserSettings`. Záznamy týchto entín sú na sebe nezávislé a tvoria celú lokálnu databázu. Pre účely tejto aplikácie je takto jednoduchá databáza dostačujúca a implementačne výhodná. Nad týmito záznamami sú vykonávané databázové dopyty, ktorých výsledky sú zobrazované do jednotlivých `Views` grafického rozhrania.

Entita `Record` reprezentujúca záznam o športovej aktivite s atribútami:

- `Sport` - povinný atribút určujúci šport ku ktorému daný záznam patrí
- `Date` - povinný atribút určujúci dátum vykonávania športovej aktivity
- `Time` - povinný atribút určujúci počet minút vykonávania športovej aktivity
- `Location` - voliteľný atribút obsahujúci miesto tréningu
- `Notes` - voliteľný atribút obsahujúci používateľové poznámky o zázname

Entita `UserSettings` reprezentujúca informácie o používateľovi:

- `Name` - povinný atribút nesúci meno používateľa
- `Surname` - povinný atribút nesúci priezvisko používateľa
- `ID` - povinný atribút obsahujúci jednoznačný identifikátor používateľa

Entita `UserSettings` bude vytvorená pri prvom spustení aplikácie novým používateľom a nesie informáciu o existencii používateľa v systéme. Zabezpečuje práve jedno ID pre každého používateľa.

Pri lokálnej databáze je však nutné zabezpečiť synchronizáciu dát s inými zariadeniami. Tento proces umožňuje automaticky trieda `NSCloudKitPersistentContainer`, za predpokladu existujúceho platného iCloud účtu.

3.5.2 Návrh online databázy

Pre správu dát online databázy je využívaný framework CloudKit (2.4.3). Online databázu predstavujú záznamy jednej jedinej entity - `UserStats`. Databázu si je možné predstaviť ako tabuľku obsahujúcu jeden záznam `UserStats` pre každého používateľa aplikácie. Tento záznam je pravidelne aktualizovaný na základe aktivity používateľov.

Motiváciou pre vytvorenie online databázy a entity `UserStats` je možnosť sledovania športových aktivít iných používateľov. Pre tento účel teda bolo nutné zdieľať dáta medzi používateľmi online pomocou služby iCloud. `UserStats` obsahuje informácie o športovej aktivite za posledný týždeň.

Entita `UserStats` reprezentujúca štatistiky o aktivite používateľa:

- `ID` - povinný atribút určujúci jednoznačný identifikátor používateľa
- `Name` - povinný atribút obsahujúci meno používateľa
- `Surname` - povinný atribút obsahujúci priezvisko používateľa
- `FavSport` - povinný atribút obsahujúci šport, ktorému používateľ venoval najviac času (za posledných 7 dní)
- `TimeOfFavSport` - povinný atribút určujúci počet minút vykonávania športu uloženého v atribúte `FavSport` (za posledných 7 dní)
- `TimeOfAllSports` - povinný atribút obsahujúci počet minút vykonávania športových aktivít zaznamenaných v aplikácii (za posledných 7 dní)
- `LastUpdate` - povinný atribút obsahujúci dátum poslednej aktualizácie daného záznamu

Záznamy tejto entity sa lokálne neukladajú, a tak je nutné uskutočniť vždy nový databázový dopyt na iCloud pri prístupe k nim.

Kapitola 4

Implementácia

Táto kapitola popisuje implementáciu aplikácie Tajmer. Sú v nej popísané predovšetkým zaujímavé či dôležité časti implementácie a predpokladá predchádzajúcu znalosť čitateľa v odvetví programovania iOS aplikácií. Nepopisuje implementáciu základných častí ako `ViewController` či `UINavigationController`, taktiež nevysvetľuje vstavané metódy riadiace životný cyklus aplikácie. Kapitola sa zameriava na implementačné detaily práce s databázami, widgetu, Siri asistentky, kontextového menu a podobne. Cieľom nie je vysvetliť podrobne celú implementáciu aplikácie, ale poskytnúť prehľad o použitých postupoch so zameraním na kľúčové časti, ktoré aplikácia obsahuje.

4.1 Základné informácie

Aplikácia Tajmer je natívna iOS aplikácia vyvíjaná v prostredí XCode 12.4 v jazyku Swift. Pri vývoji sa dodržuje vývojový vzor Model-View-Controller poskytujúci prehľadný zdrojový kód a zároveň možnosť upravovať jednotlivé časti nezávisle.

Pre prácu s lokálnou databázou je využitý framework CoreData UserDefaults. Aplikácia obsahuje 3 rozšírenia: widget, kontextové menu a podporu hlasovej asistentky Siri. Aplikácia podporuje zdieľanie dát medzi používateľmi využívajúc framework CloudKit. Autentifikácia používateľa prebieha pomocou konta iCloud. Pri implementácii grafického rozhrania boli využité frameworky UIKit a SwiftUI. Okrem natívnych frameworkov boli využité aj frameworky tretích strán: Charts a FSCalendar.

Aplikácia je vyvíjaná pre systém iOS 14.0 a vyššie, nakoľko podpora widgetov prichádza až v tejto verzii.

4.2 Grafické rozhranie

Grafické rozhranie aplikácie je vytvorené pomocou Interface Builder v prostredí XCode a frameworku UIKit. Implementácia grafického rozhrania vychádza z jeho návrhu popísaného v kapitole 3.4. Základným riadiacim komponentom je `UITabBarController`, ktorý umožňuje prepínanie medzi 5 hlavnými obrazovkami. Každá obrazovka je tvorená zo štandardných grafických prvkov ako napríklad `UIButton, UILabel, UITableView, UIPicker, UIView` a podobne. Ich pomocou je ďalej možné presmerovanie na iné obrazovky (okrem hlavných) a to 2 spôsobami. Jedným zo spôsobov je `UINavigationViewController`, ktorý funguje na princípe zásobníku, teda je možné skladať jednotlivé okná na seba a spätne sa k nim vracať využívajúc akcie `Segue` typu `Show`. Druhým spôsobom je využitie prechodu

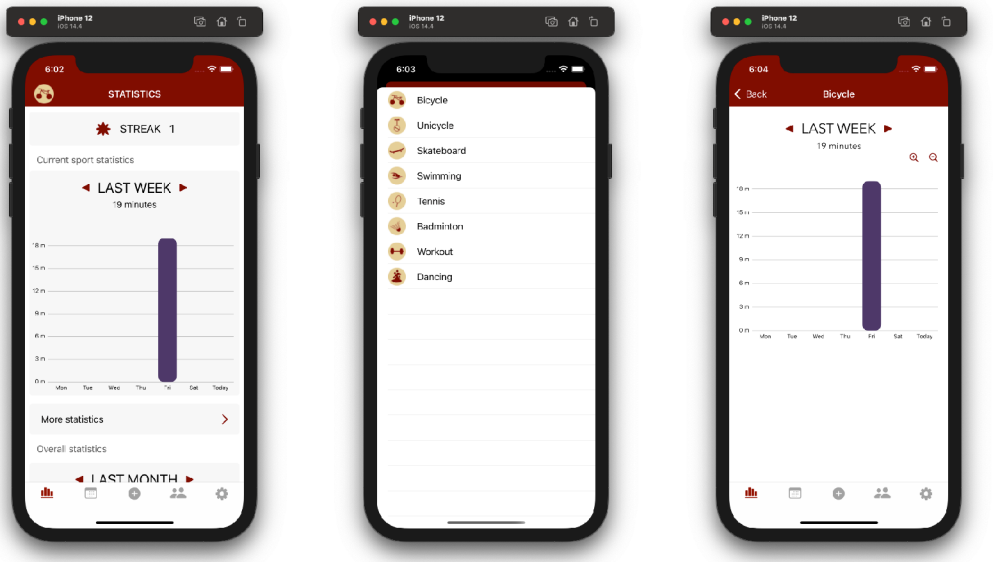
`Segue` typu `Show Detail` (bez využitia `UINavigationViewController`), ktorý zobrazí danú obrazovku tzv. modálne (viď 4.1 - obrázok vstrede).

Pre zvýšenie používateľského pohodlia pri ovládaní aplikácie, sa na každej hlavnej obrazovke nachádza horná lišta s nadpisom a `UIBarButtonItem`, ktorý umožňuje prepínanie aktuálneho športu.

Obrazovka s grafmi a prehľadom o štatistikách je tvorená pomocou `UIScrollView`, čo umožňuje vytvorenie obrazovky schopnej zobraziť obsah dlhší ako je ona sama - používateľ sa pomocou potiahnutia posúva po obsahu tejto obrazovky.

Zobrazenie športových záznamov (v rámci akejkoľvek obrazovky, ktorá to umožňuje), zoznamu používateľov, športov, polohy, nastavení a podrobných štatistík je umožnené pomocou grafického prvu `UITableView`.

Na implementáciu grafov bol využitý framework Charts a vytvorenie obrazovky s interaktívnym kalendárom prebehlo pomocou frameworku FSCalendar.



Obr. 4.1: Ukážka štruktúry grafického rozhrania aplikácie

4.3 Autentifikácia a identifikácia používateľa

Autentifikácia používateľa prebieha automaticky v závislosti na účte iCloud. Aplikácia teda vyžaduje platné konto iCloud, ktoré umožňuje dve základné funkcie. Prvou funkciou je synchronizácia dát medzi rôznymi zariadeniami jedného používateľa a druhou je zautomatizovanie vytvorenia respektíve prihlasovania sa do používateľského účtu aplikácie.

O synchronizáciu dát medzi zariadeniami sa stará trieda `NSCloudKitPersistentContainer`. V tomto prípade nie je nutné implementovať autentifikáciu používateľa manuálne, nakoľko sa oň plnohodnotne a automaticky stará framework CloudKit. Ak používateľ nemá platné iCloud konto alebo nie je prihlásený, aplikácia nedokáže synchronizovať jeho dáta medzi viacerými zariadeniami a v prípade, že je aplikácia vymazaná zo zariadenia, používateľ stráca všetky dáta.

Druhou funkciou, pre ktorú je nutné autentifikovať používateľa je vytvorenie konta v aplikácii. Takéto konto je reprezentované záznamom `UserSettings` v databáze. Konto nesie

informácie o mene, priezvisku a id používateľa. Na základe tohto záznamu je ďalej vytvorený verejný profil používateľa vo verejnej databáze ako záznam typu `UserStats`. Pre dosiahnutie čo najväčšej používateľskej prívetivosti je konto vytvorené automaticky pri prvom spustení aplikácie s prístupom na internet, pričom meno a priezvisko je získané práve z iCloud konta. Používateľ je však povinný umožniť aplikácii prístup k týmto informáciam. Vďaka tomuto prístupu je možné používateľa (jeho záznam) vo verejnej databáze jasne identifikovať.

V prípade, že používateľ už aplikáciu v minulosti používal, alebo ju začína používať na viacerých zariadeniach súčasne, autentifikácia pomocou iCloud konta výrazne zjednodušuje obnovenie či synchronizáciu jeho dát. V oboch prípadoch sa pri prvom spustení aplikácie nájde záznam `UserSettings` v privátnej iCloud databáze, ktorý obsahuje informácie o konte používateľa - predovšetkým je dôležité ID. Ak by sa tento záznam v databáze nenachádzal, jedná sa o prvú inštaláciu pre dané iCloud konto a dochádza k vytvoreniu nového záznamu `UserSettings`. iCloud teda umožní nielen obnoviť všetky záznamy, nakoľko po vymazaní aplikácie sa stále nachádzajú v cloudovom úložisku, ale zabezpečí, že sa nevytvorí viac ako jeden záznam typu `UserSettings`.

4.4 Siri

Pomocou frameworkov `Intents` a `IntentsUI` je implementovaná podpora hlasovej asistentky Siri. Vytvorenie skratky, na ktorú dokáže Siri reagovať pozostáva z niekoľkých krokov:

- vytvorenie `NSUserActivity`
- vytvorenie `INShortcut` z `NSUserActivity`
- prezentácia `INUIAddVoiceShortcutViewController`, ktorý sprostredkúva používateľské rozhranie pre vytvorenie skratky

Pomocou `INUIAddVoiceShortcutViewController` (viď obrázok 4.2 vľavo) je možné nastaviť skratku pre daný príkaz a uložiť. Ak je skratka vytvorená, vzniká objekt `INVoiceShortcut` a pomocou protokolu `INUIAddVoiceShortcutViewControllerDelegate` je nutné uložiť identifikátor tejto skratky. Vďaka identifikátoru skratky je ďalej možné skratku upraviť, či ju zmazať.

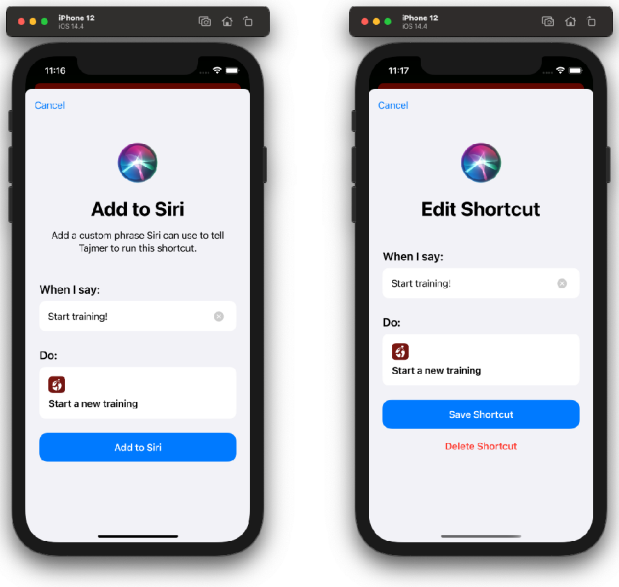
ID skratky je uložené pomocou rozhrania `UserDefaults`. V prípade, že sa používateľ rozhodne skratku upraviť či zmazať, na základe ID je prezentovaný `INUIEditVoiceShortcutViewController` (viď obrázok 4.2 vpravo), ktorý tieto akcie umožňuje. Ďalej pomocou protokolu `INUIEditVoiceShortcutViewControllerDelegate` je daná zmena vykonaná - v prípade zmazania je ID odstránené, a tak je znova možné skratku nanovo vytvoriť.

Po vytvorení skratky je však ešte nutné definovať akciu, ktorú má daná skratka vykonávať. Ich implementácia sa nachádza v triede `SceneDelegate` pomocou metódy `scene(_ scene: UIScene, continue userActivity: NSUserActivity)`. Táto metóda je schopná odlíšiť volanie jednotlivých skratiek na základe ich typu `activityType`, ktorý bol definovaný pri vytvorení `NSUserActivity` pre danú skratku. Nasledujúca časť kódu zobrazuje príklad použitia tejto metódy.

```
func scene(_ scene: UIScene, continue userActivity: NSUserActivity)
{
    switch userActivity.activityType {

    case (activityType 1):
    // vykonaj ľubovolné operácie

    case (activityType 2):
    ...
}
```



Obr. 4.2: INUIAddVoiceShortcutViewController a INUIEditVoiceShortcutViewController

4.5 Kontextové menu

Implementácia kontextového menu je sprostredkovaná pomocou triedy `UIApplicationShortcutItem`. Prostredie Xcode umožňuje definovať pole `UIApplicationShortcutItems` v súbore `Info.plist`. Definícia skratiek, ktoré sa zobrazia v kontextovom menu sa nachádza v tomto poli. Každá skratka sa skladá z definície nasledujúcich 3 objektov:

- `UIApplicationShortcutItemIconType` - ikona skratky
- `UIApplicationShortcutItemIconTitle` - nadpis skratky
- `UIApplicationShortcutItemType` - ID skratky

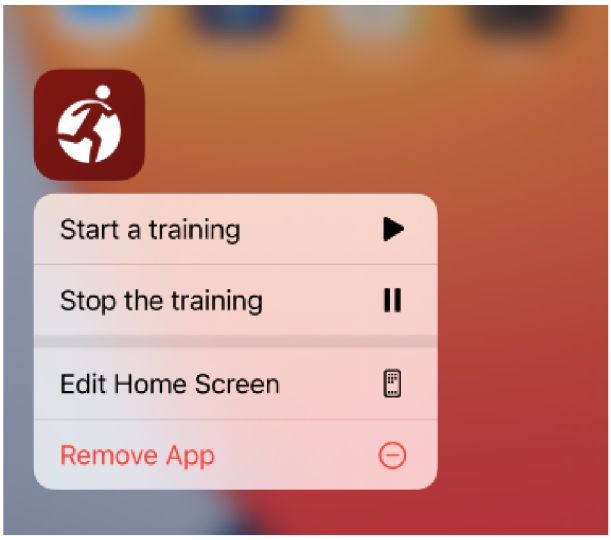
Skratkám je potrebné definovať akcie, ktoré sa majú vykonať po ich spustení. Tie sú implementované v metóde triedy `SceneDelegate - windowScene(_:performActionFor: completionHandler:)`. Jednotlivé skratky je možné identifikovať na základe ich `UIApplicationShortcutItemType`. Princíp implementácie tejto metódy je znázornený nižšie.

```
func windowScene(_ windowScene: UIWindowScene,
                 performActionFor shortcutItem: UIApplicationShortcutItem,
                 completionHandler: @escaping (Bool) -> Void)
{
    switch shortcutItem.type {

    case (UIApplicationShortcutItemType 1):
    // vykonaj ľubovolné operácie

    case (UIApplicationShortcutItemType 2):
    ...
}
```

Kontextové menu aplikácie je zobrazené na Obrázku 4.3.



Obr. 4.3: Ukážka kontextového menu aplikácie

4.6 Widget

Framework WidgetKit umožňuje jednoduchú implementáciu rôznych widgetov pre aplikáciu. Tomuto frameworku sekunduje framework SwiftUI, ktorý je jedinou možnosťou ako implementovať grafické rozhranie widgetu.

Prvým krokom implementácie je pridanie rozšírenia Widget do projektu Xcode. Po pridaní je automaticky vygenerovaný súbor .swift obsahujúci nasledujúce štruktúry (podriaďujúce sa daným protokolom), ktoré je nutné implementovať pre správnu funkciu widgetu.

- `struct SimpleEntry: TimelineEntry`
- `struct Provider: TimelineProvider`
- `struct TrainingWidgetEntryView : View`
- `struct TrainingWidget: Widget`

4.6.1 struct SimpleEntry: TimelineEntry

`SimpleEntry` predstavuje štruktúru, ktorá špecifikuje čas, kedy má byť widget aktualizovaný a súčasne dáta, ktoré budú zobrazené. Timeline entries sú tvorené v rámci `TimelineProvider`, pri vytváraní `Timeline`.

Nasledujúca časť kódu zobrazuje timeline entry štruktúru pre widget aplikácie Tajmer.

```
struct SimpleEntry: TimelineEntry {
    let date: Date
    let running: Bool
    let actualSport: Int
    let reset: Bool
}
```

4.6.2 struct Provider: TimelineProvider

Štruktúra `Provider` predstavuje kľúčovú časť pri implementácii widgetu. Definuje tri metódy vychádzajúce z protokolu `TimelineProvider`.

- `placeholder()`
- `getSnapshot()`
- `getTimeline()`

V prípade, že je widget zobrazený prvýkrát, je zobrazený jeho placeholder. Placeholder zobrazuje základný vzhľad widgetu, pričom jeho úlohou je poskytnúť používateľovi základný prehľad o tom, ako widget bude vyzerať. Pre zobrazenie placeholderu widgetu, využíva WidgetKit metódu `placeholder()`. Táto metóda vracia práve jedno timeline entry, nie celú `Timeline`, na základe ktorej rendruje jeho vzhľad.

WidgetKit volá metódu `getSnapshot()` vtedy, keď je widget zobrazený vo widget galérii. Cieľom snapshotu je zobraziť widget tak rýchlo, ako je to možné, často s príkladovými dátami. V prípade aplikácie Tajmer je placeholder a snapshot identický.

Najdôležitejšou metódou tejto podkapitoly je však `getTimeline()`. Táto metóda poskytuje pole timeline entries pomocou `Timeline`, ktoré hovoria WidgetKit-u kedy a ako aktualizovať widget.

V prípade aplikácie Tajmer `Timeline` obsahuje len jeden timeline entry a teda pomocou `Timeline` nedochádza k pravidelnej aktualizácii widgetu. Pre tento účel slúži `WidgetCenter` popísaný ďalej v podkapitole 4.6.5.

Ukážka implementácie metódy `getTimeline()`.

```
func getTimeline(in context: Context,
      completion: @escaping (Timeline<Entry>) -> ())
{

    var entries: [SimpleEntry] = []

    // získanie dát pre timeline entry
    ...

    // vytvorenie entry na základe získaných dát
    let entry = SimpleEntry(date: ..., running: ...,
                            actualSport: ..., reset: ...)
    entries.append(entry)

    let timeline = Timeline(entries: entries, policy: .atEnd)
    completion(timeline)
}
```

Na získanie dát pre timeline entry je použité rozhranie UserDefaults. Jeho zdieľanie respektíve prístup k nemu je pre aplikáciu a rozšírenie widget možné pomocou definovaného spoločného AppGroup v nastaveniach projektu aplikácie a rozšírenia. V aplikácii sú pomocou UserDefaults uložené potrebné informácie a vo widgete sa pomocou UserDefaults k týmto informáciam pristupuje. V tomto prípade sa jedná o informácie: stav prebiehajúceho tréningu a aktuálny šport.

4.6.3 struct TrainingWidgetEntryView : View

Táto štruktúra obsahuje implementáciu grafického rozhrania widgetu. Jedným z jej členov je práve štruktúra `SimpleEntry`, ktorá umožňuje prenášať dáta do widgetu, vychádzajúc z princípu popísaného v podkapitole 4.6.2.

Príklad implementácie štruktúry `TrainingWidgetEntryView`.

```
struct TrainingWidgetEntryView : View
{
    // štruktúra SimpleEntry obsahujúca dáta z aktuálneho timeline entry,
    // ktorý definuje Timeline
    var entry: Provider.Entry

    var body: some View {
        // implementácia View
    }
}
```

4.6.4 struct TrainingWidget: Widget

Konfiguráciu a obsah widgetu riadi protokol `Widget`. V tejto štruktúre je nutné nakonfigurovať widget tak, aby mu zodpovedal, respektíve ho riadil práve jeden `Provider` a obsahoval

práve jedno grafické rozhranie - `View`. Okrem toho je však možné nastaviť widgetu meno, popis, či podporované veľkosti widgetu.

Konfigurácia widgetu aplikácie Tajmer.

```
struct TrainingWidget: Widget
{
    let kind: String = "TrainingWidget"

    var body: some WidgetConfiguration {
        StaticConfiguration(kind: kind, provider: Provider()) { entry in
            TrainingWidgetEntryView(entry: entry)
        }
        .configurationDisplayName("Tajmer widget")
        .description("Widget makes using Tajmer simple.")
        .supportedFamilies([.systemMedium])
    }
}
```

4.6.5 WidgetCenter

Pre aktualizáciu obsahu widgetu je využitá trieda `WidgetCenter` - konkrétne metóda `WidgetCenter.shared.reloadAllTimelines()`. Táto metóda po zavolaní znovu načíta všetky `Timeline` definované vo WidgetKit-e. Nakoľko je v aplikácii využitá jedna jediná `Timeline`, volanie tejto metódy je dostatočné - netreba definovať konkrétnu. Aktualizácia prebieha tak, že WidgetKit žiada o `Timeline`, a tak metóda `getTimeline()` vytvára novú `Timeline` s aktuálnymi dátami.

Pred volaním `WidgetCenter.shared.reloadAllTimelines()` je nutné aktualizovať obsah v UserDefaults, nakoľko týmto spôsobom sú dáta prenášané medzi aplikáciou a widgetom.

4.6.6 Tlačidlá v GUI widgetu

Zaujímavou časťou implementácie grafického rozhrania je vytvorenie tlačídiel na widgete. Trieda `Button` nemôže byť vo widgete využitá nakoľko interaktívne prvky nie sú priamo podporované. Pre dosiahnutie „button“ efektu je v tomto prípade možné využiť štruktúru `Link`. Vo `View` widgetu je možné definovať rôzny počet štruktúr `Link`, ktoré umožňujú presmerovanie na určitú URL adresu. Jedinou podmienkou pre využitie `Link` vo widgete je veľkosť widgetu - v tomto prípade musí byť widget `.systemMedium` alebo `.systemLarge`.

Aby daný `Link` bol schopný pomocou URL otvoriť aplikáciu a vykonať určitú funkciu, je nutné pre aplikáciu definovať takzvaný `URL Type` (schému URL) - toto je možné vykonať v nastaveniach projektu. To znamená, že ak prebieha presmerovanie na URL adresu s danou schémou, aplikácia registruje tento podnet a postará sa o jeho spracovanie. Podobný princíp sa využíva pre webstránky, ktoré majú svoje aplikácie - po zadaní URL do internetového prehliadaču je otvorená aplikácia namiesto webu.

Implementácia tlačídla pomocou štruktúry `Link` vyzerá nasledovne, pričom definovaná URL schéma je training.

```
Link("  ", destination: URL(string: "training://choose")!)
    .background(Image(...)
    .resizable().scaledToFill()
    .frame(width: 45, height: 45, alignment: .center))
```

URL okrem schémy (training) však môže obsahovať aj doménu (choose), ktorá umožní identifikovať akciu, respektíve tlačídlo, ktoré bolo stlačené. Po stlačení tlačídla je možné zachytiť správu o vyslanom URL pomocou metódy `scene(_ scene: UIScene, openURLContexts URLContexts: Set<UIOpenURLContext>)` triedy `SceneDelegate` napríklad takto:

```
func scene(_ scene: UIScene,
           openURLContexts URLContexts: Set<UIOpenURLContext>)
{
    if let url = URLContexts.first?.url
    {
        let key = url.absoluteString
        // odlíšenie jednotlivých URL
        if key.contains("choose"){
            // vykonaj ľubovoľnú akciu
        }
        else if key.contains("reset") {
            // vykonaj ľubovoľnú akciu
        }
        else if key.contains("start") {
            // vykonaj ľubovoľnú akciu
        }
    }
}
```

4.7 Využitie CoreData

Pre správu lokálnej databázy bol použitý natívny framework CoreData. CoreData zabezpečuje perzistenciu používateľových nastavení `UserStats` a záznamov o športovej aktivite `Record`. O synchronizáciu týchto dát sa stará automaticky trieda `NSPersistentCloudKitContainer`.

Dôležitou časťou implementácie je práca s kontextami CoreData kontajnerov. Kontext predstavuje obraz aktuálneho stavu databázy, pričom pomocou neho je možné nové záznamy vkladať, mazať či upravovať. Základným a súčasne hlavným kontextom je takzvaný `viewContext` kontajneru. Tento kontext existuje na hlavnom vlákne aplikácie a je vytvorený automaticky pri inicializácii `NSPersistentCloudKitContainer`. Nakoľko je tento kontext prepojený s hlavným vláknom aplikácie znamená to, že akákoľvek práca s týmto kontextom bude počas svojej aktivity blokovať hlavné vlákno. V prípade, že aplikácia bude obsahovať veľký počet záznamov a vykonávať náročné operácie nad nimi, existencia jediného kontextu môže spôsobiť, že bude aplikácia neresponzívna a grafické prostredie bude mrznúť. Preto je vhodné popri hlavnom kontexte pracovať s `backgroundContext` a hlavný kontext využívať výlučne na operácie spojené s používateľským prostredím.

Funkcia `newBackgroundContext()` vytvorí aktuálny obraz (kontext) databázy na vedľajšom vlákne pre daný kontajner. Ďalej je práca s takýmto kontextom v podstate identická

ako s hlavným. Je možné záznamy vkladať, mazať či upravovať. Takýto kontext je priamo prepojený s `NSPersistentStoreCoordinator`, čo znamená nezávislé úpravy databázy. Pre synchronizáciu vedľajšieho kontextu s hlavným je preto nutné nastaviť takzvanú *property* hlavného kontextu `.automaticallyMergesChangesFromParent` na *true*. Toto však nie je nutné v aplikácii Tajmer, nakoľko vedľajší kontext slúži výlučne na tvorenie databázových dopytov (ang. fetch request) na pozadí aplikácie. Konkrétne sa jedná o situáciu, kedy je nutné získať štatistiky, ktoré sú ukladané do záznamu `UserStats`. Tým, že tento dopyt prebieha pomocou vedľajšieho kontextu, je tento proces nezávislý od plynulosti používateľského prostredia.

4.7.1 NSFetchedResultsController

K vykonaniu databázového dopytu, ktorého výsledky sú zobrazené do používateľského prostredia, je využitá trieda `NSFetchedResultsController`. Táto trieda totiž umožňuje implementačne veľmi jednoduchú automatickú aktualizáciu grafického rozhrania v prípade, že sa dáta v databáze zmenia. Pomocou protokolu `NSFetchedResultsControllerDelegate` je možné zachytávať zmeny v databáze, ktoré ovplyvnia daný dopyt. Takýto dopyt sa uskutoční len raz. To znamená, že ak používateľ pridá/aktualizuje/vymaže záznam, `NSFetchedResultsControllerDelegate` túto skutočnosť zachytí a je možné zareagovať na každú z týchto správ jednotlivo.

`NSFetchedResultsController` je využitý pre obrazovku obsahujúcu grafy a kalendár, pre zabezpečenie vždy aktuálneho grafického rozhrania. Ak je zmenený aktuálny šport v aplikácii, je nutné zmeniť dopyt vo `NSFetchedResultsController`, a tak znova prebehne inicializácia `NSFetchedResultsController` pomocou nového dopytu.

4.8 Využitie UserDefaults

Trieda `UserDefaults` je využitá na niekoľko účelov, ktoré vyžadujú perzistentnosť nejakej jednoduchej informácie. Konkrétne sa jedná napríklad o detekeciu prvého spustenia aplikácie, uloženie informácie o poslednej aktualizácii verejného záznamu ale aj uloženie ID Siri skratiek (viac v podkapitole 4.4). Okrem toho zohráva dôležitú úlohu pri prenášaní informácií medzi aplikáciou a widgetom (viď 4.6.2).

Vďaka jednoduchosti tejto triedy je ju vhodné využiť namiesto frameworku Core Data v opísaných prípadoch. Súčasne `UserDefaults` slúži výlučne na zabezpečenie lokálnej perzistencie dát, a tak rozlišuje jednotlivé používateľove zariadenia. Táto skutočnosť je dôležitá predovšetkým v prípade detekcie prvého spustenia aplikácie, nakoľko pri prvom spustení na novom zariadení je vždy nutné autentifikovať používateľa.

4.9 Zdieľanie dát medzi používateľmi

Zdieľanie dát medzi používateľmi spostredkuje verejná databáza uložená na úložisku iCloud vytvorená pomocou frameworku CloudKit. Pri prvom prihlásení používateľa - respektíve prvom spustení aplikácie sa vytvorí záznam triedy `CKRecord` typu `UserStats`, obsahujúci informácie o používateľovi a jeho týždenný prehľad o aktivite (viac v 3.5.2). Jednému používateľovi prináleží práve jeden záznam, ktorý je možné jednoznačne identifikovať pomocou ID používateľa. Raz za deň prebehne aktualizácia tohto záznamu. Ak používateľ nainštaluje aplikáciu po druhýkrát na jednom zo svojich zariadení, prebehne databázový dopyt záznamov a ak nájde záznam so svojím ID, nový záznam `UserStats` nie je vytvorený.

Implementácia používateľského rozhrania, zobrazujúca časť zdieľania dát je rozdelená na dve časti. Prvou je `UIViewController` obsahujúci `UITableView` a `UISearchBar`. Pomocou searchbaru je možné filtrovať jednotlivé záznamy v databáze, a tak získať záznam žiadaného používateľa. Po kliknutí na `UITableCell` obsahujúce používateľove meno sa zobrazí druhá časť používateľského rozhrania - porovnanie medzi aktuálnym používateľom aplikácie a vyhľadaným. Táto obrazovka obsahuje jednoduchý graf a tabuľku obsahujúcu porovanie celkovej aktivity používateľov a aktivity najobľúbenejších športov. Ak záznam vyhľadávaného používateľa nie je aktuálny, používateľ je oboznámený touto skutočnosťou pomocou `UIButton` typu info. Po kliknutí na toto tlačídlo sa zobrazí `UIAlertController`, obsahujúci počet dní, od obdobia keď bol naposledy záznam aktualizovaný. Ak vyhľadaný používateľ nebol aktívny viac ako 7 dní, porovnanie sa nezobrazí a namiesto grafu sa nachádza informácia o neaktivite používateľa.

4.10 Využitie Notification Center

Predávanie informácií medzi jednotlivými `ViewController`-mi je možné viacerými spôsobmi. Vo väčšine prípadov je tento proces implementovaný pomocou tzv. segues. V aplikácii je však okrem segues podobné predávanie informácií implementované aj pomocou `NotificationCenter`. Táto trieda umožňuje posielať informácie spôsobom, kedy pozorovatelia sledujú určitú notifikáciu a ak je notifikácia zaslaná, dokážu na ňu reagovať. Tento prístup bol zvolený pri obrazovke, ktorá umožňuje výber športu. Ak používateľ zmení aktuálny šport, rozpošle sa notifikácia všetkým registrovaným pozorovateľom, čo umožní zmeniť aktuálny šport na všetkých obrazovkách aplikácie.

4.11 Získavanie polohy

Každý záznam môže niesť informáciu o polohe, kde bol tréning vykonaný. Používateľ má 2 možnosti, ako svoju polohu zadať. Buď využije GPS, alebo môže vyhľadať polohu manuálne. Na implementáciu oboch spôsobov je použitý framework `Core Location`.

Pri využití GPS je kľúčová trieda `CLLocationManager` a protokol `CLLocationManagerDelegate`. Pri inicializácii `CLLocationManager` je možné nastaviť presnosť GPS a je možné hľadať polohu zariadenia. Na jej použitie si je nutné vypýtať povolenie od používateľa. V prípade nájdenia polohy `CLLocationManagerDelegate` o nej informuje tak, že pomocou metódy `locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation])` umožní pristúpiť k poľu objektov `CLLocation`. V tomto poli sa nachádzajú súradnice určujúce možné polohy zariadenia. V ďalšom kroku je nutné z `CLLocation` získať názov, respektíve konkrétnu informáciu o štáte, meste, prípadne ulici a podobne. O túto časť sa stará trieda `CLGeocoder`, ktorá umožňuje získať pole `CLPlacemark` pomocou metódy `reverseGeocodeLocation()`. Každá `CLPlacemark` obsahuje konkrétne informácie o polohe - adresu. Takto získaná adresa je ďalej prezentovaná používateľovi v tabuľke, pomocou ktorej si môže vybrať jednu z navrhnutých.

Na implementáciu manuálneho výberu adresy je použitá databáza, ktorú obsahuje trieda `CLGeocoder`. Pomocou vyhľadávacej lišty môže používateľ zadať kľúčové slovo, ktoré je spracované a použité ako vstupný parameter pre metódu `geocodeAddressString()`. Táto metóda vracia pole objektov `CLPlacemark`. `CLPlacemark` je následne spracovaná podobne ako pri získavaní GPS a prezentovaná do tabuľky, kde si opäť používateľ môže vybrať navhodnejšiu polohu.

Kapitola 5

Testovanie

Testovanie aplikácie prebiehalo tromi spôsobmi: unit testovanie, testovanie pomocou simulátoru a testovanie na fyzickom zariadení iPhone. V nasledujúcej kapitole je popísaný proces testovania počas implementácie, ale aj finálnej aplikácie. Do testovania boli zapojení používatelia, ktorí buď hodnotili aplikáciu na základe vlastnej skúsenosti formou dotazníka, alebo bola ich práca s aplikáciou pozorovaná a následne vyhodnotená. Z výsledkov testovania, ktoré odhalili implementačné chyby, boli vyvodené následky, respektíve bola aplikácia upravená a znova zasadená do testovania. Tento proces sa opakoval, až kým sa v testovaní neprejavili nedostatky.

5.1 Unit testovanie

Na overenie správnosti implementácie jednotlivých častí aplikácie bolo použité takzvané unit testovanie. Unit testovanie umožňuje nezávisle testovať rôzne časti zdrojového kódu, čo predchádza chybám počas implementácie - respektíve ich objavovaniu. Pri testovaní funkcie sa pomocou unit testu porovnáva jej výstup a očakávaný výsledok. V prípade, že sa výsledky zhodujú, je možné test prehlásiť ako úspešný. Unit testovanie bolo vykonávané pravidelne počas implementácie nových funkcií, či ich dodatočnej úpravy.

Testovacia časť je nezávislá časť projektu, teda nijako neovplyvňuje zdrojový kód aplikácie a je ju možné spustiť samostatne. Na tieto účely bol využitý natívny framework XCTest v prostredí XCode. Pomocou triedy `XCTestCase` je definovaná sada testov, ktorú je možné spustiť ako celok ale aj jednotlivo. Ak testovanie neprebehlo úspešne (aspoň jeden test zlyhal), chyba musela byť opravená pred pokračovaním v implementácii, inak by testovanie strácalo zmysel.

5.2 Testovanie na simulátore

Počas implementácie aplikácie bol hlavným prostriedkom pre testovanie jednotlivých častí simulátor v prostredí Xcode. Tento simulátor je schopný plnohodnotne simulovať prostredie akéhokoľvek Apple zariadenia s operačným systémom iOS, watchOS, tvOS a iPadOS - v tomto prípade sa ale jedná o iOS. Ako dve hlavné zariadenia boli vybraté: iPhone 12 a iPhone 8 s operačným systémom iOS 14.4.

Testovanie pozostávalo z viacerých častí:

- Testovanie GUI - jednotlivých grafických komponent

- Testovanie správnosti implementácie ViewControllerov
- Testovanie synchronizácie dát
- Testovanie aplikácie ako celku

Pri pridávaní akéhokoľvek grafického prvku do používateľského rozhrania bola jeho funkcionalita dôkladne testovaná. Ku kľúčovým bodom tohto testovania patrí napríklad testovanie interakcií gestami, správne rozloženie vo view, prechody medzi jednotlivými obrazovkami či responzivita grafického rozhrania.

Spojenie modelovej a view časti aplikácie sa implementuje pomocou takzvaných ViewControllerov. Pomocou grafického rozhrania je preto vhodné ich správnu funkcionalitu skontrolovať respektíve testovať. Príkladom pre takýto test je získavanie dát, ktoré sú zobrazené napríklad do `UITableView`. Pomocou simulátora je možné overiť, či `UITableView` obsahuje očakávané dáta.

Simulátor umožňuje taktiež spustenie viacerých zariadení súčasne, a tak poskytuje veľmi pohodlné prostredie na testovanie synchronizácie dát medzi viacerými zariadeniami.

Keď aplikácia vytvorila funkčný celok, prebehla komplexným testovaním, kedy bolo nutné overiť funkčnosť jednotlivých obrazoviek, databáz, ich prepojenie pričom sa jednalo o simuláciu reálneho použitia aplikácie.

V prípade, že mala aplikácia v simulátore neočakávané správanie, bolo nutné túto chybu odstrániť a testy opakovať. Nie je rozumné sa však na simulátor naplno spoliehať, nakoľko je možné, že sa na ňom niektoré chyby neprejavia. Taktiež simulátor nepodporuje niektoré funkcie ako napr. telefonovanie, otváranie e-mailového klienta a podobne. Z tohto dôvodu je vhodné zvoliť aj inú formu testovania - testovanie na fyzickom zariadení (5.3).

5.3 Testovanie na fyzickom zariadení

Pre testovacie účely bol ako fyzické zariadenie použitý iPhone SE 2. Testovanie na fyzickom zariadení prebiehalo pravidelne, avšak vo väčších intervaloch ako testovanie na simulátore. Cieľom tohto testovania bolo predovšetkým odhaliť chyby a testovať funkcie, ktoré buď simulátor nepreukázal, alebo nedokáže testovať. Takéto testovanie však umožnilo získať reálnu skúsenosť s aplikáciou, testovať ju s pripojením ale aj bez pripojenia na internet (komplikované cez simulátor), či jednoducho zobrať aplikáciu „do terénu“.

5.4 Testovanie s používateľmi

V nasledujúcej kapitole je popísané testovanie, ktorého sa zúčastnili reálni používatelia. Testovanie s využitím používateľov prebehlo dvomi spôsobmi: 1.) Moderované osobné testovanie (5.4.1) 2.) Dlhodobé testovanie (5.4.2).

5.4.1 Moderované osobné testovanie

V prípade moderovaného testovania sa jedná o testy, ktoré prebiehali tak, že používateľom bol zadaný zoznam určitých úloh, ktoré mali splniť. Pri vypracovávaní týchto úloh boli sledovaní a na základe ich reakcií boli pokladané dodatočné úlohy či otázky. Na základe tohto testovania bolo možné sledovať rôzne aspekty aplikácie a robiť si o testovaní poznámky, vďaka ktorým mohla byť aplikácia dodatočne upravená. Cieľom tohto testovania

bolo predovšetkým sledovať, ako je používateľ schopný s aplikáciou pracovať - intuitívnosť grafického prostredia.

Používatelia, ktorí sa zúčastnili testovania, nemali žiadnu predošlú skúsenosť s aplikáciou. Každý test pozostával z nasledujúcich úloh:

1. Spustiť/zastaviť stopky a z nameraného času vytvoriť záznam (aktuálny deň, poloha) pre šport plávanie
2. Vytvoriť dodatočný záznam (ľubovoľný čas, dátum, poloha)
3. Upraviť záznam z bodu 1.) tak, že bude modifikovaný čas a dátum
4. Zmazať ľubovolný záznam
5. Zobraziť štatistiky posledného mesiaca pre ľubovoľný šport a upraviť jeden zo záznamov mimo rozhrania kalendáru
6. Zobraziť verejné štatistiky používateľa Ondrej Kondek
7. Nakonfigurovať ľubovoľný príkaz Siri a otestovať ho
8. Pridať widget a jeho pomocou zopakovať bod 1.)
9. Spustiť a zastaviť tréning pomocou kontextového menu

Testovanie vykonávali piati respondenti, pričom úlohy vykonávali v uvedenom poradí. Jeden z respondentov uviedol, že pri prvej úlohe, nebolo jasné ako vytvorený záznam uložiť a musel na to prísť vylučovacou metódou. Vylučovacia metóda v tomto prípade nie je ideálna, nakoľko môže trvať dlhší čas, kým sa používateľ s aplikáciou zoznámi. Faktom však je, že pri ostatných používateľoch sa tento problém neprejavil, plynulo postupovali pri plnení úlohy. Pri úlohách s poradovým číslo 2 až 4 a 6 až 9 používatelia postupovali bez akýchkoľvek zaváhaní.

Jediným problémom bola úloha číslo 5. V tomto prípade bolo nutné kliknúť na graf a v grafe na stĺpec, pomocou ktorého je možné záznamy zobraziť a upraviť. Zobraziť štatistiky za posledný mesiac nebol problém, ďalej však používatelia mali tendenciu skúsiť ostatné možnosti na tejto obrazovke, nie kliknúť na graf. Nasledovalo kliknutie na graf, v tomto prípade očakávali už výsledok a teda zobrazenie príslušných záznamov. Tu bolo nutné ešte označiť príslušný stĺpec v grafe, ostatné už bolo jednoznačné. Ak sa používateľ dostal do rozhrania grafu, netrvalo mu dlho pochopiť, že musí ešte označiť stĺpec. Nebolo to však správanie aplikácie, ktoré by používatelia očakávali.

Všetky úlohy používatelia dokázali vyriešiť v rámci jednotiek sekúnd, v prípade problematickej úlohy 1 a 5 maximálne 10-20 sekúnd. Na základe toho neboli v rámci grafického rozhrania vykonané žiadne zmeny.

Počas testovania intuitívnosti aplikácie sa však prejavili drobné chyby v aplikácii, či nedostatky, ktoré narušovali komfort používateľa. Napríklad po uložení záznamu sa časovač nevynuloval, a tak pri spustení časovač pokračoval v starom tréningu. Na základe diskusie s používateľmi tak boli po testovaní vykonané drobné zmeny, ktoré buď opravili implementačné nedostatky, alebo vylepšili samotné prostredie pre používateľov.

5.4.2 Dlhodobé testovanie

Dlhodobé testovanie prebiehalo približne mesiac tak, že traja používatelia aktívne využívali aplikáciu Tajmer na svojich vlastných zariadeniach. Pri tomto testovaní bolo simulované reálne využitie aplikácie. Vďaka mesačnému testovaniu mohli používatelia vyskúšať a dôkladne preskúmať funkcionalitu aplikácie, jej grafické rozhranie a taktiež rozšírenia, ktoré aplikácia obsahuje. Okrem toho bola nepriamo testovaná kvalita implementácie, nakoľko sa mohli pri používaní objaviť problémy, či nedostatky, ktoré sa pri krátkodobom testovaní neprejavili. Počas testovania používatelia pravidelne podávali spätnú väzbu, na základe ktorej bola aplikácia upravovaná. Následne bola aktuálna verzia aplikácie znova nahraná do používateľských zariadení. Posledná verzia aplikácie bola nahraná dva týždne pred ukončením testovania, aby bolo výsledky testovania možné vyhodnotiť.

Po skončení testovania každý používateľ vyplnil dotazník (viď príloha B). Na základe tohto dotazníku bolo možné testovanie vyhodnotiť. Nasledujúca tabuľka obsahuje priemerné hodnotenie aplikácie používateľmi po mesačnom testovaní.

Funkcionalita	Intuitívnosť	Dizajn	Celkový dojem	Odporúčanie
4.55/5	4.3/5	4.3/5	4,65/5	100%

Tabuľka 5.1: Výsledky testovania

Okrem vyhodnotenia vo forme tabuľky dotazník obsahoval niekoľko častí, ktoré napomohli finálnemu vylepšeniu aplikácie. Používatelia mali možnosť nahlásiť rôzne chyby, ktoré sa prejavili a navrhnúť vylepšenia grafického rozhrania. Súčasne bolo odsledované používanie alternatívnych spôsobov zadávania dát, ich využitie, intuitívnosť a podobne.

Respondenti neuviedli žiadne neočakávané správanie, ktoré by bránilo plynulému chodu aplikácie. Jedinou nahlásenou chybou bol problém s aktivitou, ktorá prebiehala počas viacerých dní, čo viedlo k nameraniu nesprávneho času. Na základe toho bola daná chyba odstránená.

Grafické rozhranie aplikácie vyhovovalo väčšine používateľov - nemali žiadne výhrady. Dvaja respondenti uviedli, že intuitívnosť aplikácie, a tak aj ich hodnotenie bolo výrazne znížené skutočnosťou, že uloženie aktuálnej aktivity (čas nameraný pomocou stopiek) pre nich bolo mätúce. Nakoľko sa tento problém vyskytol aj pri moderovanom testovaní, do grafického rozhrania bolo pridané tlačídlo, ktoré slúži na uloženie aktuálne nameraného tréningu.

Pri testovaní rozšírení aplikácie - widget, Siri a kontextové menu - respondenti nespozorovali žiadne problémy. Používateľom dané možnosti pomáhali pri zaznamenávaní aktivít a ich rozhranie hodnotia ako dostatočne intuitívne. Taktiež uviedli, že apoň jedno rozšírenie aktívne používali, a tak je možné usúdiť, že aplikačné rozšírenia v tomto prípade dávaju zmysel z pohľadu používateľského komfortu.

Poslednou časťou dotazníku je celkové hodnotenie aplikácie. Používatelia na aplikácii najviac ocenili možnosť pridávať spätne záznamy o aktivitách a naopak vytkli nemožnosť zmeny aktivity priamo v rozhraní, kedy dochádza k potvrdeniu uloženia záznamu. 100% používateľov by aplikáciu odporučilo známym a taktiež by si ju nainštalovalo v prípade, že bude uverejnená na AppStore. Celkový dojem z aplikácie bol priemerne 4.3/5, a tak je možné usúdiť, že aplikácia má potenciál a používatelia sú s ňou spokojní - takže jej návrh a implementácia prebehli úspešne.

5.5 Vyhodnotenie testovania

Počas vývoja aplikácie prebiehalo pravidelne niekoľko testovaní: unit testovanie, testovanie pomocou simulátoru a na fyzickom zariadení. Toto testovanie som vykonával sám, aby som overil funkčnosť buď jednotlivých častí implementácie alebo menších celkov aplikácie. Opakované vykonávanie testov zaručilo stabilnú verziu aplikácie a umožnilo overovať správnosť implementácie postupne.

Keď bola dokončená prvá verzia aplikácie - teda aplikácia bola schopná fungovať ako celok na zariadení a spĺňala svoju navrhnutú funkcionalitu, prišlo na rad používateľské testovanie. Úlohou tohto testovania bolo dokázať, že aplikácia je správne navrhnutá, implementovaná a že spĺňa všetky používateľské potreby. Hlavným bodom tohto testovania bolo sledovanie intuitívnosti grafického rozhrania.

Pri používateľskom testovaní moderovanou formou, nemali používatelia zavážný problém s plnením úloh, a tak je možné usúdiť, že grafické rozhranie aplikácie je navrhnuté správne - respektíve zodpovedá používateľským potrebám a nevymyká sa zaužívaným štandardom.

Počas dlhodobého používateľského testovania sa neobjavili chyby v plynulosti aplikácie, v offline či online režime alebo chyby spojené s perzistenciou dát. Drobné implementačné chyby, ako napríklad nezobrazovanie sa klávesnice boli riešené operatívne, tak že používatelia dostávali obratom aktualizované verzie. Toto testovanie ponúklo používateľom dostatočný čas na zoznámenie sa s aplikáciou, s jej funkciami a rozšíreniami. Vďaka tomu bolo možné získať podrobnejšiu spätnú väzbu týkajúcu sa nielen grafického rozhrania, ale aj použiteľnosti aplikácie. V dotazníku respondenti uviedli, že aplikácia má podľa ich názoru potenciál uspieť, takúto aplikáciu by aktívne používali a taktiež by ju odporučili známym.

Pri oboch používateľských testovaniach sa vyskytol problém s funckiou uloženia nameraného záznamu. Keďže sa tento problém objavil viackrát, pristúpilo sa k dodatočnej úprave grafického rozhrania aplikácie. Funkcia pôvodného tlačidla, ktoré uloženie umožňovalo, bola upravená tak, že teraz zabezpečuje ukladanie výlučne spätných záznamov. Pre uloženie aktuálne nameraného záznamu bolo pridané celkom nové tlačidlo.

Pri jednotlivých testovaniach boli odhalené drobné chyby, ktoré vyžadovali opravu. Po každej takejto oprave boli opätovne vykonané unit testy a testovanie na simulátore. Chyby objavené počas moderovaného testovania už však neviedli k opakovaniu tohto testovania kvôli jeho časovej náročnosti. Počas dlhodobého testovania boli nedostatky priebežne opravované a aplikácia aktualizovaná.

Vďaka testovaniu bola aplikácia vylepšená a následne bolo preukázané, že aplikácia je funkčná, používateľsky prívetivá a spĺňa všetky kritéria, ktoré boli stanovené v jej návrhu. Na základe toho je možné vývoj tejto verzie aplikácie vyhlásiť ako finálny a úspešný.

Kapitola 6

Záver

Predmetom tejto bakalárskej práce bolo preskúmať nástroje a technológie používané na vývoj iOS aplikácií a súčasne návrh a implementácia vlastnej aplikácie.

Najprv som vykonal a dôkladne popísal rozbor existujúcich nástrojov používaných na vývoj iOS aplikácií ako je vývojové prostredie XCode, jazyk Swift, frameworky CoreData, CloudKit alebo frameworky tretích strán FS Calendar či Charts. Ďalej som na základe prieskumu konkurečných riešení navrhol a implementoval aplikáciu Tajmer využívajúc znalostí z predchádzajúceho prieskumu nástrojov. Úlohou tejto aplikácie je zaznamenávanie športových aktivít používateľa, ich uloženie, zobrazenie štatistík ale aj zdieľanie dát medzi používateľmi. Aplikácia kladie dôraz na pohodlné zadávanie vstupných dát, pričom využíva moderné iOS technológie ako je widget, kontextové menu ale aj hlasová asistentka Siri, ktoré sú kľúčovou časťou tejto bakalárskej práce. Posledným bodom práce bolo testovanie tejto aplikácie pozostávajúce z viacerých typov testovania - unit testovanie, testovanie pomocou simulátora a reálnych používateľov.

Pomocou testovania bola aplikácia dôkladne skontrolovaná a na základe spätnej väzby používateľov je možné usúdiť, že je aplikácia funkčná, použiteľná a napĺňa potreby, ktoré používateľ vyžaduje od daného typu aplikácie.

Aplikácia však taktiež necháva otvorený priestor prípadnej aktualizácii, kde by mohli byť pridané rôzne vylepšenia. Ako vhodné vylepšenia je možné uviesť napríklad podporu tmavého režimu, súkromné používanie (možnosť nastaviť zdieľanie dát), alebo používateľská úprava widgetu, či pokročilejšia asistencia Siri. Pri vylepšeniach týkajúcich sa widgetu a Siri je však potrebné byť vlastníkom plného vývojárskeho účtu Apple, nakoľko študentské konto tieto možnosti obmedzuje. Okrem uvedených vylepšení by však bolo vhodné aplikáciu taktiež vytvoriť pre operačný systém Android, prípadne macOS, Windows a podobne, nakoľko takto je použitie aplikácie obmedzené výlučne pre používateľov iOS. V tomto prípade by však bolo vhodné zvážiť použitie multiplatformových frameworkov ako napríklad Realm pre správu dát.

Literatúra

[1] Apple. *About Swift* [online]. [cit. 2021-20-01]. Dostupné z: https://swift.org/about/.

[2] Apple. *CloudKit* [online]. [cit. 2021-20-01]. Dostupné z: https://developer.apple.com/documentation/cloudkit.

[3] Apple. *Core Data Stack* [online]. [cit. 2021-24-02]. Dostupné z: https://developer.apple.com/documentation/coredata/core_data_stack.

[4] Apple. *UserDefaults* [online]. [cit. 2021-26-01]. Dostupné z: https://developer.apple.com/documentation/foundation/userdefaults.

[5] Apple. *What is Core Data* [online]. [cit. 2021-26-01]. Dostupné z: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/CoreData/index.html.

[6] Apple. *WidgetKit* [online]. [cit. 2021-20-04]. Dostupné z: https://developer.apple.com/documentation/widgetkit/.

[7] Apple. *Model-View-Controller* [online]. 2018 [cit. 2021-20-01]. Dostupné z: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html.

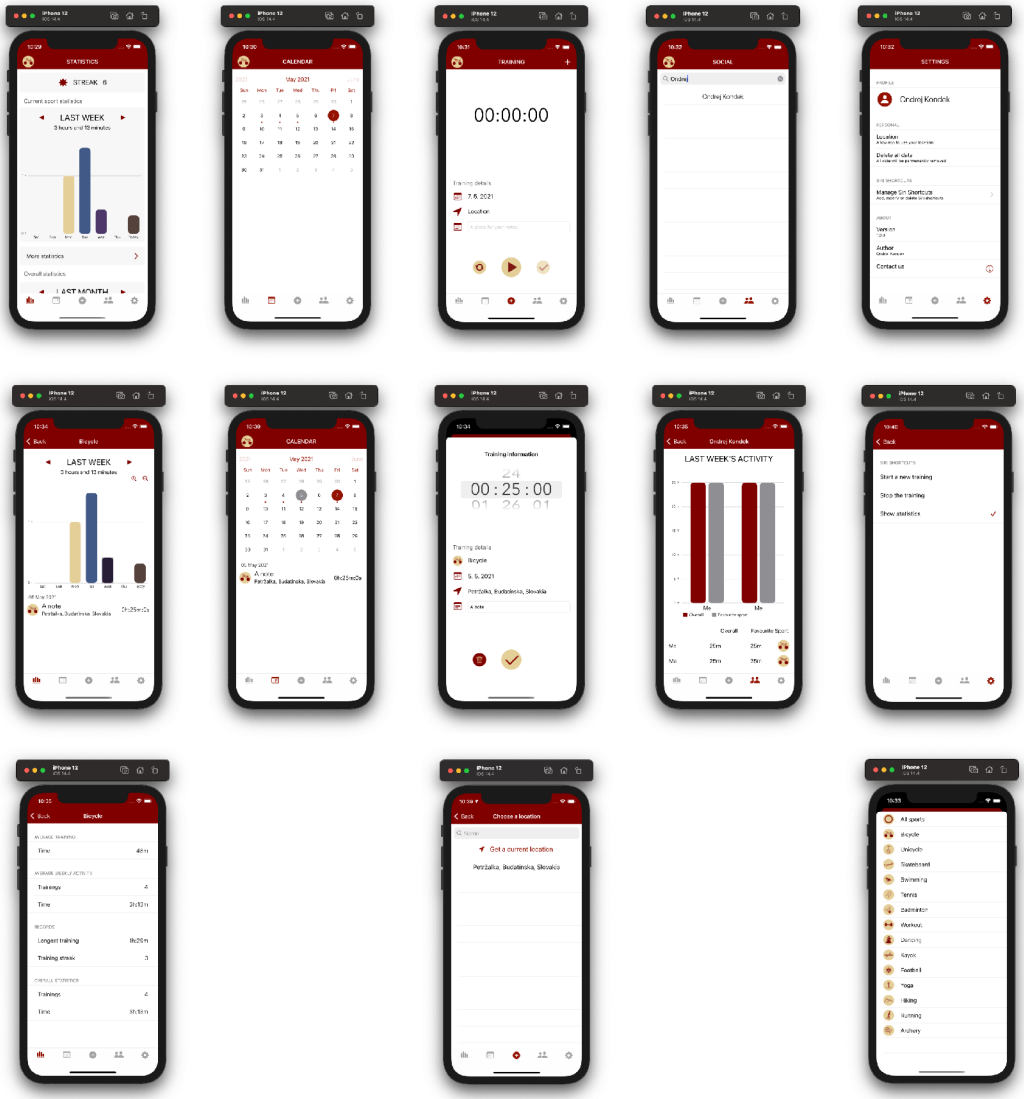
[8] Delaney, B. *Strava Live turns smartphone into real-time Strava computer* [online]. Immediate Media Company Limited, 16. júla 2015 [cit. 2021-24-02]. Dostupné z: https://www.bikeradar.com/news/strava-live-turns-smartphone-into-real-time-strava-computer/.

[9] Guyot, A. *The New Fitness App in iOS 14* [online]. MacStories, 3. septembra 2020 [cit. 2021-24-02]. Dostupné z: https://www.macstories.net/stories/the-new-fitness-app-in-ios-14/.

[10] Therox, O. a team, C. *FSCalendar* [online]. 2017 [cit. 2021-16-02]. Dostupné z: http://cocoadocs.org/docsets/FSCalendar/2.7.9/.

Príloha A

Ukážka aplikácie Tajmer



Obr. A.1: Ukážka aplikácie Tajmer

Príloha B

Dotazník

Nasledujúca príloha obsahuje dotazník, pomocou ktorého bolo vyhodnotené dlhodobé testovanie aplikácie. Obsahuje 4 časti: testovanie implementácie, grafické rozhranie aplikácie, alternatívne zadávanie vstupných dát a celkové hodnotenie aplikácie.

Testovanie implementácie

1. Počas testovania sa prejavila chyba neresponzívneho grafického rozhrania (sekanie, spomalenie, a pod.)
   Áno, často / Niekedy áno / Veľmi zriedkavo / Nevšimol som si túto chybu
2. Počas testovania sa vyskytly chyby s uložením dát, ich úpravou, či mazaním
   Áno, často / Niekedy áno / Veľmi zriedkavo / Nevšimol som si túto chybu
3. Aplikácia v online aj offline režime fungovala bez problémov.
   Súhlasím - nezbadal som rozdiel / Nesúhlasím - pri offline sa prejavili určité nedostatky / Nesúhlasím - pri online sa - prejavili určité nedostatky
4. Ak ste zbadali nedostatky v offline/online režime - popíšte chybu
5. Pri používaní widgetu alebo Siri sa vyskytli nedostatky
   Všetko bolo v poriadku / Stretol som sa s určitou chybou vo widgete / Stretol som sa s určitou chybou pri komunikácii so Siri
6. Prosím, popíšte chybu, ktorú ste zaregistrovali pri používaní Siri alebo widgetu
7. Počas testovania sa prejavila iná chyba:

Grafické rozhranie aplikácie

1. Ako by ste zhodnotili intuitívnosť grafického rozhrania (GUI)?
   Všetko v poriadku, nemám výhrady / GUI niekedy pôsobilo mätúco, ale v rámci normy (s výhradami) / GUI pôsobilo neintuitívne - určité funkcie som dlho musel hľadať / GUI je nevhodné - nepodarilo sa mi nájsť niektoré z funkcií
2. Ohodnoťte intuitívnosť grafického rozhrania známkou 1-5
3. Ako hodnotíte grafické rozhranie z pohľadu dizajnu? 1-5

4. Čo by ste zmenili na používateľskom prostredí?

5. Popíšte nedostatok, ktorý robil GUI neprehľadným/nepoužiteľným... (napr. tlačídlo spustiť tréning je príliš malé)

Alternatívne zadávanie vstupných dát

1. Ako hodnotíte formy zadávania dát do aplikácie? (widget, kontextové menu, siri)
   Aktívne som využíval tieto možnosti (aspoň jednu z nich) / Občas som si pomohol využitím jednej z možností / Takéto zadávanie dát mi nevyhovuje

2. Ktoré z daných možností pre vstup dát ste používali najviac?
   Siri / Widget / Kontextové menu / Klasickú obrazovku v aplikácii

3. Je rozhranie widgetu dostatočne intuitívne?
   Áno, je jasné, na čo slúži na prvý pohľad / Je v poriadku, no musel som jeho funkcie testovať / Vôbec nebolo jasné na čo widget slúži / Nepochopil som na čo slúži

4. Siri reaguje na 3 podnety - spusiť/zastaviť tréning, ukázať štatistiky... Akú funkciu by ste pridali?

Celkové hodnotenie aplikácie

1. Ako hodnotíte celkový dojem z aplikácie? 1-5

2. Ako hodnotíte využitie aplikácie Tajmer?
   Aplikácia má potenciál, nájde miesto na trhu / Aplikácia má potenciál, avšak potrebuje čiastočne upraviť či doplniť funkcionalitu / Aplikácia vyžaduje kľúčové zmeny pre koncový trh / Aplikácia je nezmysel

3. Odporučili by ste používanie aplikácie Tajmer? 1-5

4. Stiahli by ste si a používali by ste aplikáciu po jej uverejnení na AppStore?
   Určite áno - aktívne / Áno - raz za čas / Áno - no nepoužíval by som ju takmer vôbec / Nie - nevidím v tom význam

5. Čo vás na aplikácii najviac zaujalo narozdiel od konkurencie?

6. Čo sa vám na aplikácii nepáčilo?

7. Sem napíšte návrhy na vylepšenia: