Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра Информатики

Дисциплина: Программирование (П)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

Мобильное фитнесс-приложение под IOS

Студент: гр. 653501 Бороховская К.Г.

Руководитель: ассистент кафедры информатики Козуб

Минск 2017

Содержание

Введение…………………………………………………………………………3

1. Анализ предметной области…………………………………………………4

1.1. Анализ существующих аналогов…………………………………...4

1.2. Постановка задачи…………………………………………………...9

2. Разработка программы для отслеживания питания и активности……………………………………………...……………………….10

2.1. Технологии, библиотеки и их использование в работе приложения …………………………………………………………………………….10

2.2.Требования к программе………………………………………….…12

2.3. Принцип работы программы.……………………………………….13

3.Методика работы с полученной программой………………………………14

Заключение……………………………………………………………………...17

Литература………………………………………………………………………18

Приложение 1. Исходный код………………………………………………….19

Приложение 2. Скриншоты интерфейса

Введение

*В современном мире каждому человеку важно следить за состоянием своего здоровья и внешним видом. Важными составляющими повседневной жизни являются приемы пищи и ежедневная активность. «Вы то, что вы едите» - говорит старая пословица. И ведь верно, от нашего питания зависит правильность протекания жизненных процессов в организме.*

Рассуждения о правильном питании и спорте стали лейтмотивом современности. Люди начали задумываться о состоянии своего организма, здоровье, своем образе жизни. И теперь мы знаем, что основа здорового организма это правильное питание и активный образ жизни. Трудно переоценить важность рационального питания для поддержания здоровья человека и гармонии в его жизни, внешней и внутренней красоты.

Основными принципами здорового питания состоят в следующем:

− соответствие энергетической ценности потребляемых продуктов энергетическим затратам организма;

− обязательно должен учитываться химический состав веществ, соответствующий индивидуальным особенностям человека и его физиологическим потребностям.

Действительно при соблюдении диеты нужно не только воздерживаться от потребления калорийных продуктов, способствующих накоплению избыточного веса, но и ежедневно использовать разнообразные продукты, содержащие не менее семидесяти различных веществ, которые не синтезируются в организме. Самостоятельно сделать такие расчеты очень трудно.

Разработанное приложение поможет каждому качественно отслеживать количество и качество потребляемых им продуктов, а также следить за уровнем активности. Оно написано на языке программирования Swift для мобильных продуктов Apple. А данное приложение предлагает простой, приятный глазу дизайн, интерфейс, в котором не сложно разобраться, и удобство в использовании.

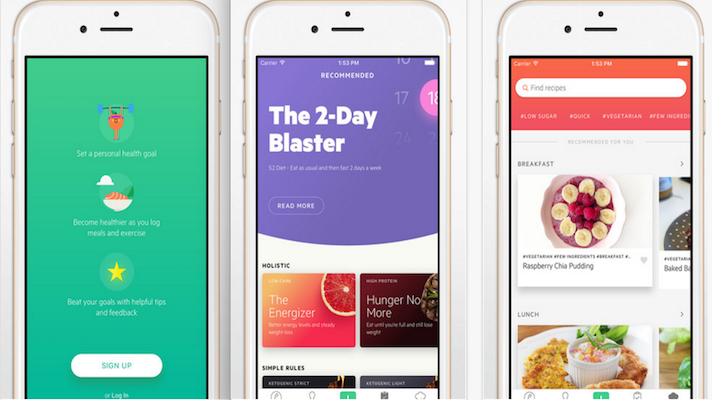
1. Анализ предметной области

1.1. Анализ существующих аналогов

Вы скажете, что существует уже много подобных приложений, и они буквально заполонили магазины мобильных программ. Чем же это приложение лучше? А тем, что здесь нет ничего лишнего, все чётко, просто, понятно и не требует много времени. Можно запутаться в разнообразии программ, предлагаемых вашими телефонами. У всех этих приложениях очень много функций: тут тебе и счётчик калорий, и социальная сеть, и обработчик фотографий, и личный тренер. Но зачем же так много всего? Это лишь отнимает у вас ваше драгоценное время. Рассмотрим следующий перечень программ: Lifesum, PumpUp, FatSecret

***Первое приложение : LifeSum.***

Интерфейс выглядит так:



Плюсы программмы:

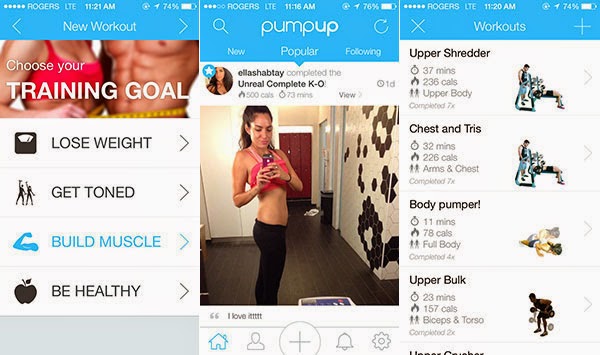
* Интересная организация и дизайн
* Множество настроек
* Персональная программа
* Огромная база данных продуктов

Минусы:

* Ограниченный функционал, доступный бесплатно, постоянно предлагает премиум-подписку за определенное количество долларов в месяц/год.
* Очень много всяких, часто ненужных, функций
* Не всегда интуитивно понятный интерфейс

***Второе приложение PumpUp.***

Интерфейс программы:



Плюсы программы:

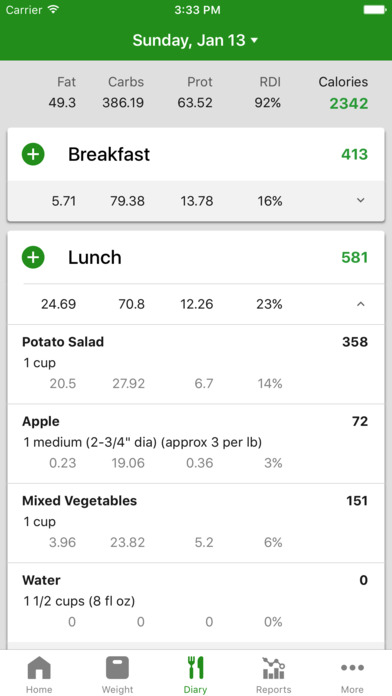
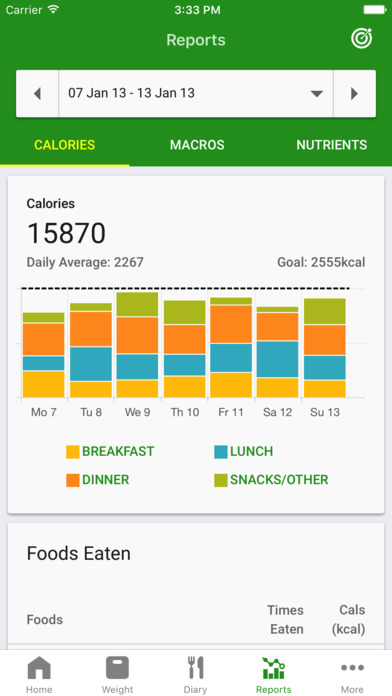
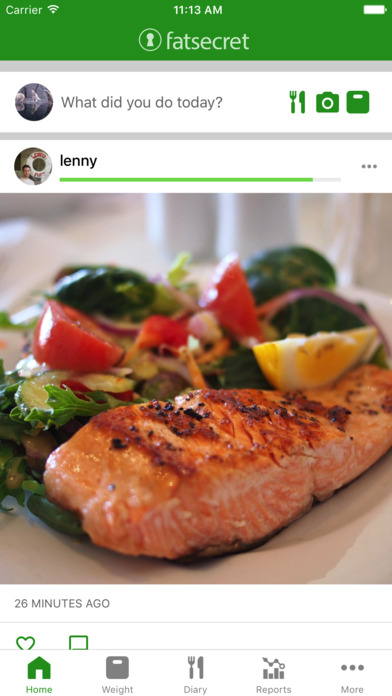
* Программа бесплатная
* Разные программы тренировок
* Социализировано
* Множество видеороликов с упражнениями

Минусы:

* Не очень интересная и удобная организация и дизайн
* Нельзя вести счет калориям

***Третье приложение FatSecret***

Интерфейс:

******

Плюсы:

* Программа бесплатная
* Большая база данных продуктов
* Наличие настроек

Минусы:

* Не всегда понятное и простое в использовании
* Иногда кажется, что слишком много всего
* Нельзя считать количество затраченных калорий

1.2 Постановка задачи

В задачу курсового проекта входит:

* Разработать программное средство на языке Swift способное выполнять подсчет количества потребляемых и затрачиваемых калорий.
* Программное средство должно выполнять следующие функции:
  + Сбор и анализ персональных данных, вывод результатов
  + Предоставлять базы данных продуктов и активностей
  + Добавление новых элементов в исходные базы данных
  + Добавление данных в базы данных каждого дня
  + Возможность просмотра данных предыдущих дней с помощью календаря
  + Возможность удаления элементов из баз данных
  + Возможность изменения персональных данных
* Реализовать понятный и удобный для пользователя интерфейс программного средства.

Таким образом, задача данного курсового проекта сводится к разработке приложения, которое поможет каждому качественно отслеживать количество и качество потребляемых им продуктов, а также следить за уровнем активности. Пользователь имеет возможность на основе своих персональных данных вести дневник своего питания и движения.

2. Разработка программы для отслеживания питания и движений

2.1 Технологии, библиотеки и их использование в работе приложения

До того как начать непосредственно разрабатывать приложение я задумалась о том, как же мне реализовать мою работу с данными в приложении, так как это является основной составляющей моей работы.

Мне нужно было разобраться как сделать так, чтобы моя программа могла:

1. Загружать базу данных продуктов и активностей, которая всегда будет доступна пользователю
2. Дать возможность добавления элементов в базу данных
3. Хранить персональные данные пользователя
4. Создавать базу данных, где храниться ежедневная информация
5. Добавлять в эту базу данных информацию о потребляемых продуктах и активностях

В конце концов я решила, что для работы с базами данных я буду использовать Realm (<https://realm.io/docs/>). Realm - это решение для кросс-платформенной мобильной базы данных, разработанное специально для мобильных приложений. Он очень удобный, быстрый, легкий, его не очень сложно интегрировать в свой проект. Наиболее часто используемые функции, такие как запросы к базе данных, пишутся буквально в пару строк.

Первоначально должен быть создан класс, в котором есть свойства хранящие данные об объекте. Например об активности. Тут у меня поля, хранящие название активности и количество калорий, затрачиваемых за час. Аналогично выглядят классы для хранения продуктов, данных о пользователе и ежедневных данных.(Исходный код, пункты 6, 8, 11, 12)

class Activity: Object {

dynamic var id = NSUUID().uuidString

dynamic var nameOfActivity = ""

dynamic var calories = 0

override class func primaryKey() -> String?{

return "id"

}

}

Так же важной составляющей экземпляра класса является индивидуальное ID. Благодаря неповторяющимся ID можно получать данные каждого экземпляра с помощью функции primaryKey. Особенно важно мне было это использовать для того, чтобы получить доступ к экземпляру базы сегодняшнего дня, где в качестве ID я использую дату сегодняшнего дня. Я создаю его при первой в этот день загрузке приложения:

// check if we already made realm element for today

if let realm = try? Realm(),

let \_ = realm.object(ofType: DailyData.self, forPrimaryKey: today as AnyObject){

// if have do nothing

} else { // if not made it

let realm = try! Realm()

let todayData = DailyData()

try! realm.write

{

realm.add(todayData)

}

}

Вот так мы получаем данные используя ID:

realm.object(ofType: DailyData.self, forPrimaryKey: today as AnyObject)

Далее я часто использую этот метод, чтобы добавлять новые элементы продуктов и активностей в эту базу данных.

Когда мне нужно получить объекты, я должны определить, какие модели я хотите получить. Таким образом сначала создается экземпляр Realm:

let realm = try! Realm()

А затем заполняется, например, foodFromBase через вызов objects(\_:), передавая имени класса тип модели, который вам нужен.

foodFromBase = realm.objects(Food.self)

Где foodFromBase является переменной типа данных Results<Food>:

var foodFromBase: Results<Food>!

На сайте Swiftbook.ru хорошо описаны концепции и основные классы Realm:

***Realm***: экземпляры Realm - это сердце фреймворка, это ваша точка доступа к основной базе данных, аналогично управляемому контексту объекта в Core Data. Вы будете создавать экземпляры при помощи инициализатора **Realm()**.

***Object*** (Объект): Это ваша модель Realm. Процесс создания модели определяет схему базы данных. Для того, чтобы создать модель вы просто создаете подкласс **Object** и определяете ***fields*** (поля), которые вы хотите сохранить как свойства.

***Relationships*** (Связи или отношения): Вы создаете связь one-to-many (один ко многим) между объектами, просто объявляя свойство типа **Object**, на которое вы хотите ссылаться. Вы можете создать связь многие-к-одному и многие-ко-многим через свойство типа **List**, что приводит вас к …

***Write Transactions*** (Запись операций): Любые операции в базе данных, такие как создание, редактирование или удаление объектов, должны выполняться в рамках операций ***writes***, которые происходят с помощью вызова **write(\_:)** у экземляров **Realm**.

***Queries*** (Запросы): Для извлечения объектов из базы данных вам нужно использовать запросы. Самая простая форма запроса - это вызов **objects()** у экземпляра **Realm**, передавая класс **Object**, который вы ищете. Если ваши поисковые запросы являются более сложными, вы можете использовать предикаты, выстраивая запросы, и также организуя результаты поиска.

***Results*** (Результаты): Результаты - это автоматически обновляемый тип контейнера, который вы получаете в ответ на объектные запросы. У них много общего с обычными массивами, в том числе синтаксис сабскрипта для захвата элемента в индексе.

После того как я получила нужную мне модель я могу получать доступ к определенному элементу: let food = foodFromBase[indexPath.row]

И работать с ним с помощью метода write:

try! foodFromBase.realm!.write {

let food = self.foodFromBase[indexPath.row]

self.foodFromBase.realm!.delete(food)

}

А для того, чтобы хранить исходные базы данных я использовала JSON. Мои изначальные базы выглядят таким образом:

{

"Food" :

{

"1" : {

"nameOfFood" : "Apple",

"calories" : 47,

"fat" : 0.4,

"carbohydrates" : 9.8,

"protein" : 0.4

},

"2" : {

"nameOfFood" : "Banana",

"calories" : 95,

"fat" : 0.2,

"carbohydrates" : 21.8,

"protein" : 1.5

},

"3" : {

"nameOfFood" : "Pinapple",

"calories" : 42,

"fat" : 0.3,

"carbohydrates" : 10.9,

"protein" : 0.4

}

}

}

}

Я не могла найти как же импортировать данные из файла json в realm. Я использовала библиотеку SwiftyJSON, чтобы импортировать json-данные в Swift, а потом уже использовала методы realm, чтобы иметь возможность в дальнейшем работать с ней. Вот эта функция, которая переводит мои данные о продуктах из json в realm:

static func getJSONAFoodData()

{

let realm = try! Realm()

if let path = Bundle.main.path(forResource: "Food", ofType: "json") { //get connection to json-file with food

do {

let data = try NSData(contentsOfFile: path, options: NSData.ReadingOptions.mappedIfSafe)

do {

let clearJSONFood = JSON(data: data as Data, options: JSONSerialization.ReadingOptions.mutableContainers, error: nil)

let numberOfFood = clearJSONFood["Food"].count

// fill realm base with Food base

for i in 0..<numberOfFood{

let food = Food()

food.nameOfFood = clearJSONFood["Food","\(i + 1)", "nameOfFood"].stringValue

food.calories = clearJSONFood["Food", "\(i + 1)", "calories"].intValue

food.fat = clearJSONFood["Food", "\(i + 1)", "fat"].doubleValue

food.carbohydrates = clearJSONFood["Food", "\(i + 1)", "carbohydrates"].doubleValue

food.protein = clearJSONFood["Food", "\(i + 1)", "protein"].doubleValue

try! realm.write

{

realm.add(food)

}

}

}

} catch {}

}

}

Аналогично выглядит функция для импортирования данных об активностях.

Далее мне было необходимо разобраться как работать с датами, как написать календарь. Я начала создавать свой календарь с помощью библиотеки JTAppleCalendar. После дня работы у меня получилось создать свой собственный дизайн календаря, после чего я поняла, что писать календарь полностью самой это не очень хорошая идея, поэтому я нашла подходящий для меня проект на githab.com и внедрила его в свой. Для того, чтобы качественно вставить его в свой проект и разобраться, каким образом он написан и как им пользоваться, я провела многие часы, работая. Для меня это стало очень ценным опытом, так как я считаю, что это мне поможет для дальнейшей совместной разработки с другими программистами.

2.2 Требования к приложению для отслеживания питания и движения

Необходимо написать программу на Swift, которая смогла бы хранить данные о затрачиваемых и потребляемых калориях пользователя с возможностью просмотра данных предыдущих дней. В связи с этим к программе предъявляются следующие требования:

1. Интерфейс.

Интерфейс программного средства будет содержать три главных экрана, между которыми вы перемещаетесь с помощью трех кнопок в нижней части экрана. Центральная кнопка отвечает за Home-экран, на котором пользователь видит данные о потраченных и употребившихся калориях. С помощью кнопки в левом углу экрана можно вызвать календарь. А с помощью кнопки в правом углу экрана можно вызвать экран с персональными данными. Нажатие на левую кнопку в нижней части экрана переносит вас на экран, где размещена база данных активностей. С помощью кнопки в правом верхнем углу экрана мы перемещаемся на экран добавления нового элемента в базу данных. Если мы нажмем на ячейку с информацией об активности, мы переместимся на экран добавления активности в дневную базу данных. Нажатие на правую кнопку в нижней части экрана переносит вас на экран, где размещена база данных продуктов. Далее по аналогии с схемой левой кнопки. Также предусмотрены 3 единично появляющиеся экраны при первой загрузке приложения. На первом размещен Welcome screen, после экран заполнения персональных данных, следом экран с результатами подсчетов.

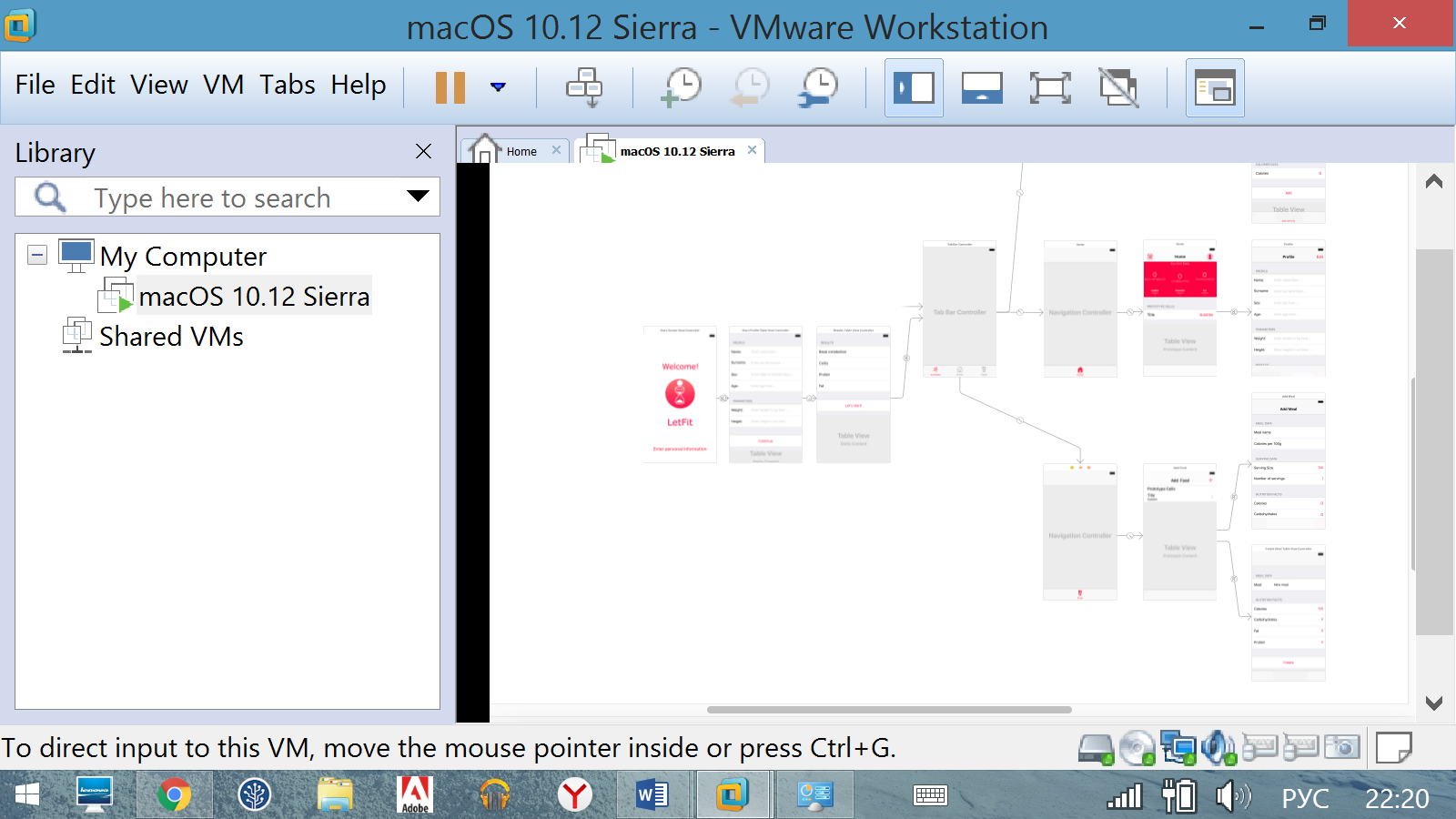
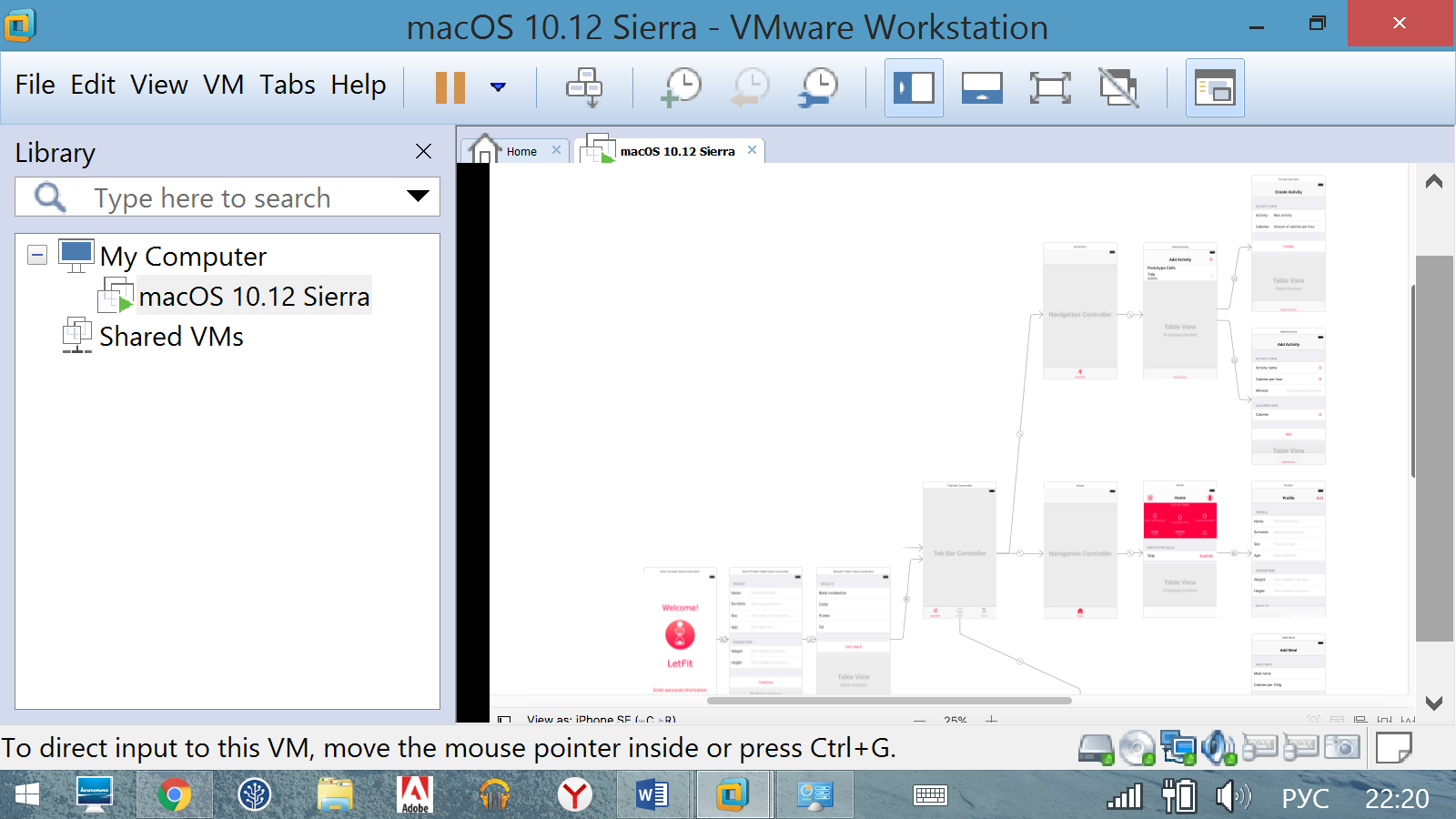
1. Функциональность.

Данная программа должна выполнять следующие функции:

* + Предоставлять пользователю выбор элементов из баз данных продуктов и активностей
  + Давать возможность добавления и удаления элементов из баз данных
  + Давать возможность просмотра добавленных данных
  + Высчитывать базовый метаболизм, количество рекомендуемых к употреблению жиров, белков и углеводах на основе персональных данных

1. Совместимость. Данное программное средство совместимо с мобильными телефонами компании Apple, у которых програмное обеспечение обновлено до версии 9.0 и выше .
2. Среда разработки. Средой разработки данного программного средства выбран Xcode. Это максимально удобная среда для разработки приложений на языках Objective-C и Swift.

2.3 Принцип работы

Это схема моего приложения:

Основная задача работы этого приложения это считывание, передача, добавление, вывод, удаление элементов информации баз данных. Моей задачей было подать пользователю этот функционал красиво, просто и удобно.

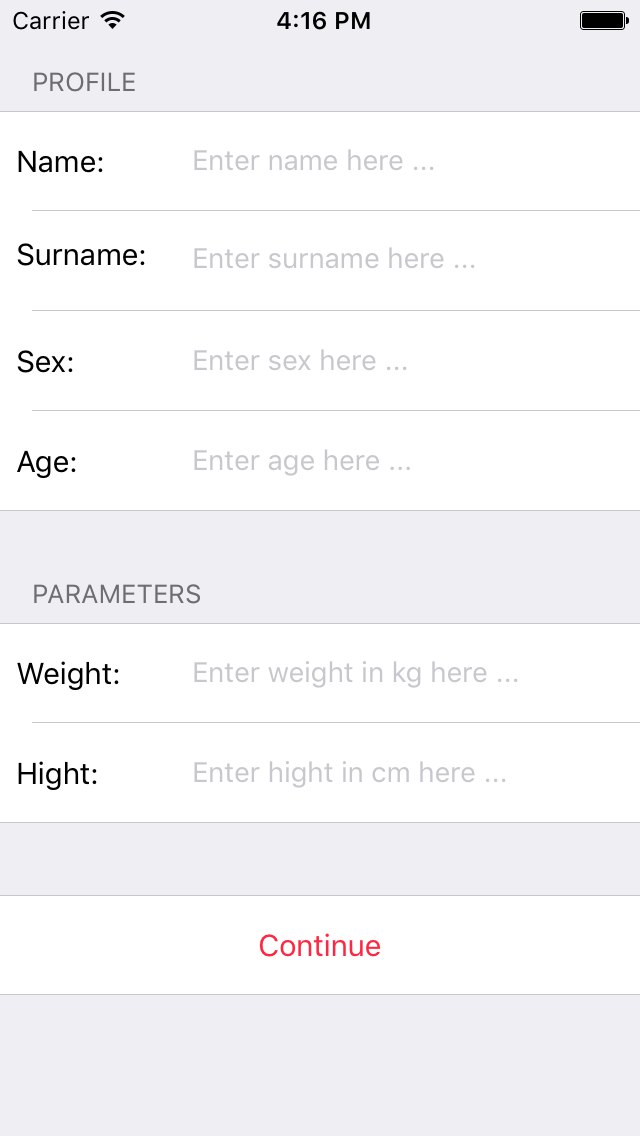
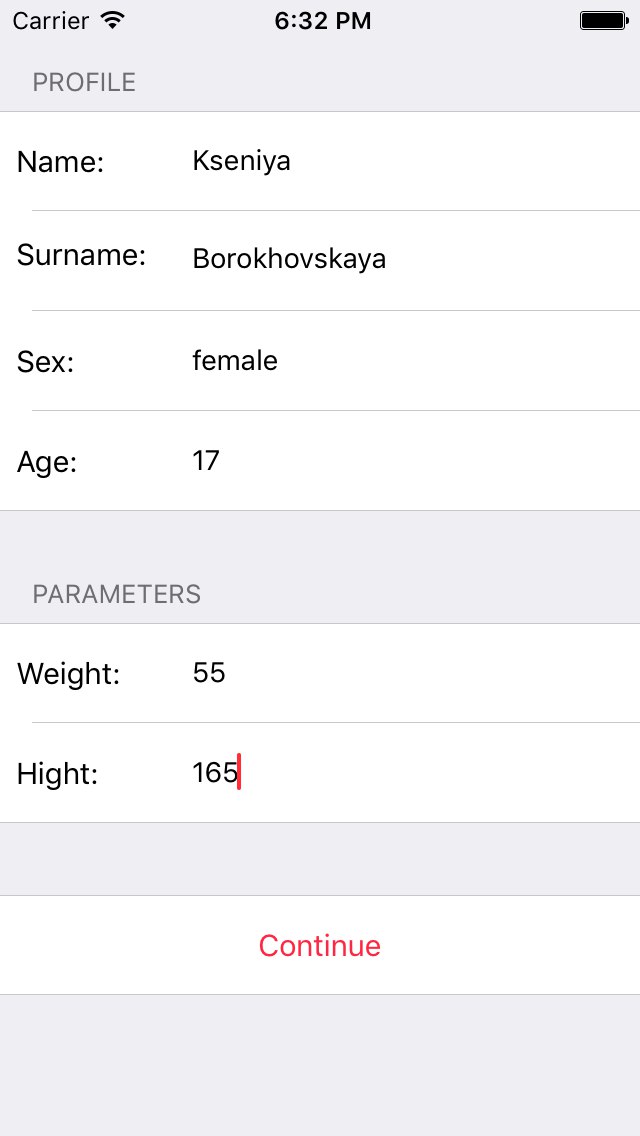
Не важно какого типа приложение создается, все приложения, так или иначе, используют следующие паттерны проектирования:

\* Model-View-Controller - Этот паттерн проектирования управляет всей структурой приложения.

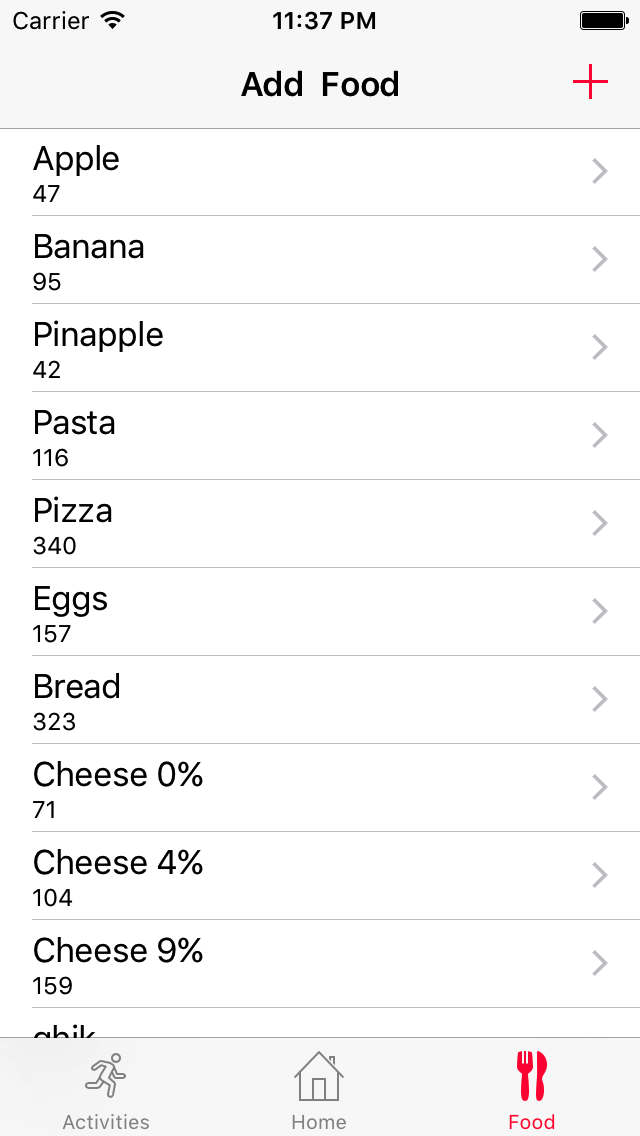
\* Delegation - Этот паттерн проектирования облегчает передачу информации от одного объекта другому.

\* Target-action - Этот паттерн проектирования переводит взаимодействие пользователя с пользовательским интерфейсом в код, который приложение может выполнить.

Так, как я уже писала раньше, для удобного манипулирования информацией из баз данных я использую классы. Это и есть мои Models. Экраны приложения – это View, к каждому из ник прикреплены файлы, контролирующие работу этого View. Это Controllers. Каждый такой файл назван соответствующе, например, AddActivityTableViewController. Аналогично названы все контроллеры моего проекта. У каждого контроллера есть жизненный цикл, задающийся вызывающимися в определенный момент функциями. Например во ViewWillAppear я говорю моему контроллеру что делать перед тем как мой экран появится.

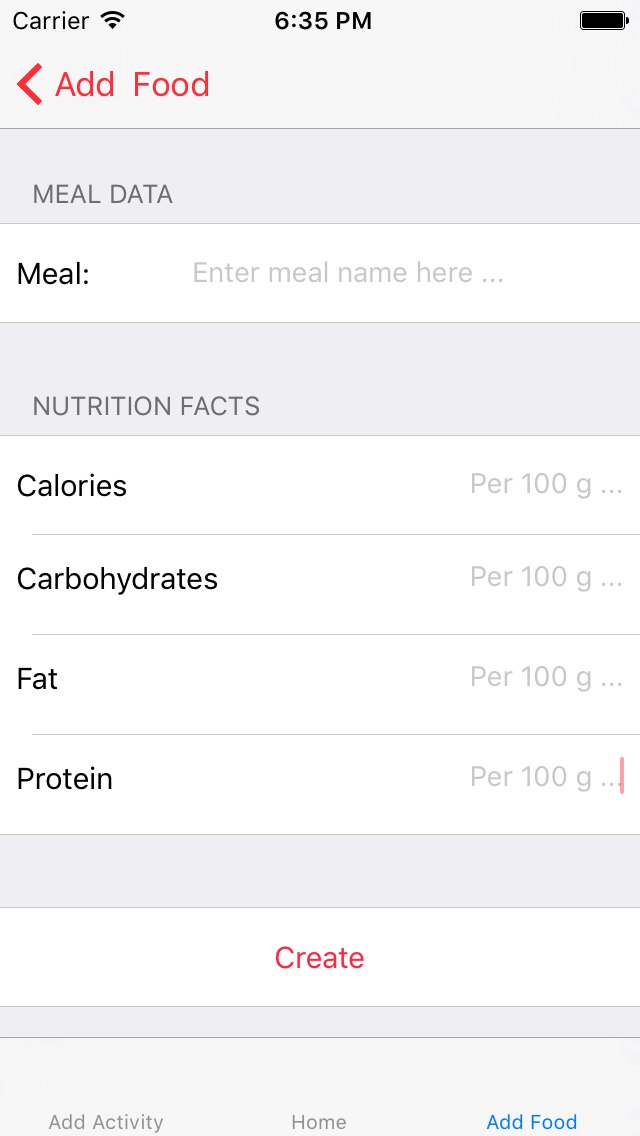


Это экран, где пользователь впервые вводит какую-либо информацию. После нажатия на кнопку Continue создается элемент класса User, где и храниться персональная информация пользователя.

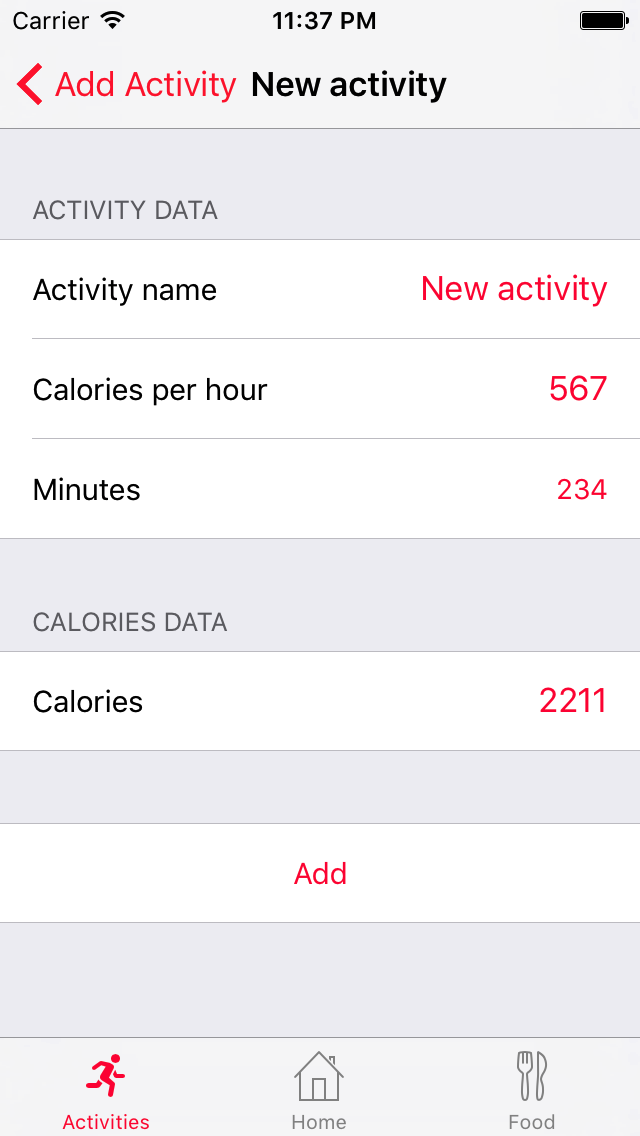


База данных, которую пользователь видит на данном экране загружается один раз при первой загрузке приложения. Это все происходит в файле AppDelegate в функции application, которая выполняется при загрузке приложения. Тут же создается экземпляр класса DailyData при первой в этот день загрузке приложения, где хранятся данные о употребившихся и затраченных в этот день калориях.

На экране ниже мы видим как можно добавить новый элемент в базы данных продуктов и активностей. При нажатии на кнопку Create новый элемент добавляется в базу.

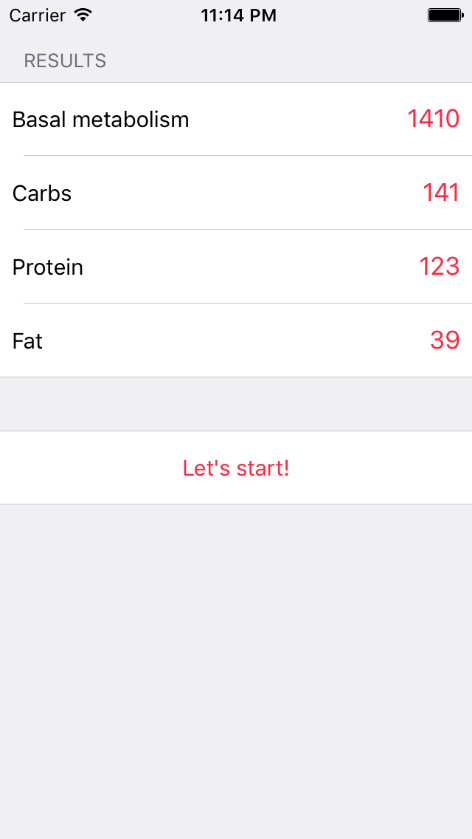
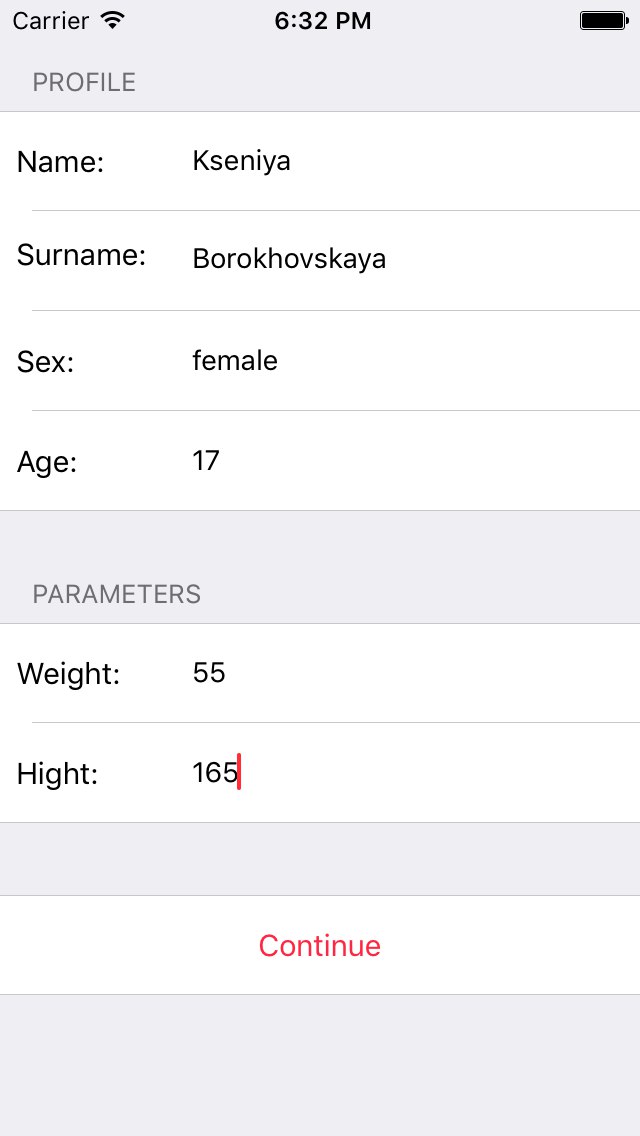
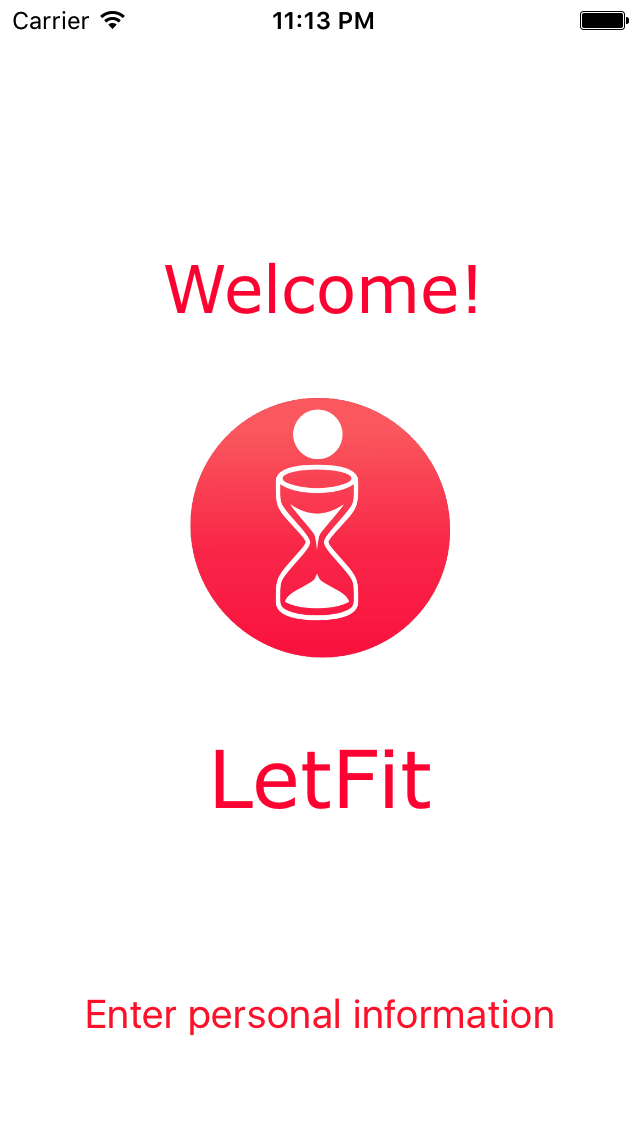


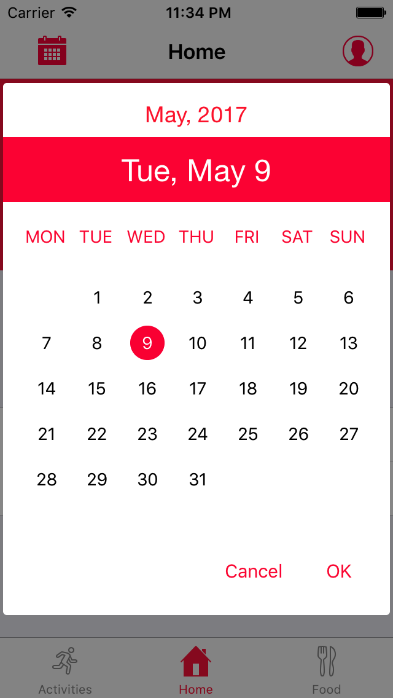
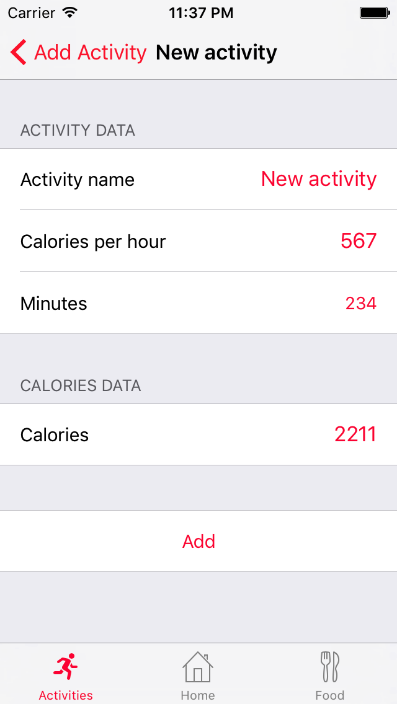
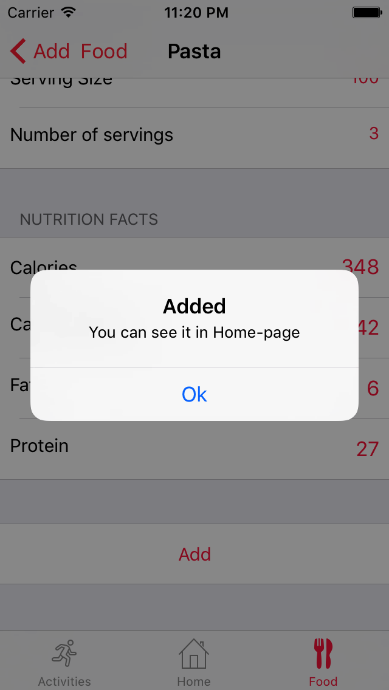
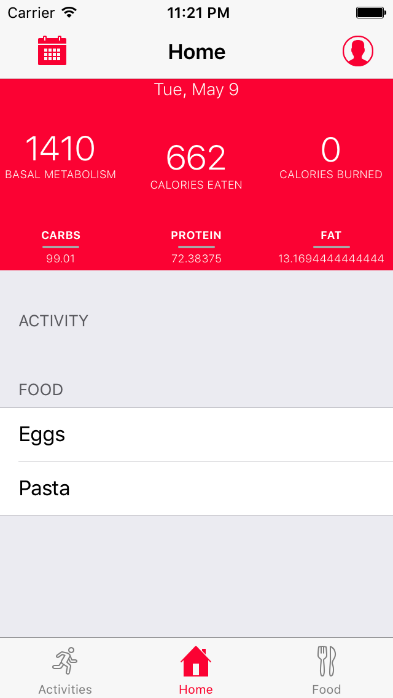
На следующем экране мы добавляем элементы в базу данных этого дня.



3. Методика работы с полученной программой

Как я уже пиала выше, впервые загружая приложение на экран телефона пользователь переходить от одного из трех экранов к другому. Вот они:



Пользователь вводит данные, получает результаты. И вот, наконец-то пользователь может работать с приложением. Теперь он может пользоваться базами данных, добавлять туда новые элементы, следить за своим образом жизни.

Заключение

В результате работы над курсовым проектом было разработано приложение для отслеживания питания и активностей.

В ходе работы над проектом был изучен язык Swift, изучены концепции разработки приложений на этом языке. Я прочитала и использовала очень много официальной документации по этому языку, прошла курс лекций Стенфордского университета, искала различные статьи и видео для того, чтобы лучше разобраться в необходимых мне темах

В конце концов я использовала библиотеки RealmSwift, SwiftyJSON, JTAppeCalendar в своем приложении для того, чтобы сделать прилодемие максимально удобным и функциональным. Все базы данных в моем приложении контролируются написанными мною классами, а каждый View контролируется отдельным файлом.

Итак, что же включает в себя разрабатываемое приложение. Впервые загружая его на экран своего телефона вы вводите своё имя и индивидуальные параметры, на основе которых программа высчитывает количество калорий, рекомендуемое для употребления в день, соотношение БЖУ, которое подходит лично вам, базовый обмен веществ (это количество энергии, которое вы расходуете каждый день без учёта активности. В соответствии с этими данными вы теперь можете вести дневник вашего питания и движения. В самой программе есть база данных продуктов с данными об их калорийности и БЖУ, а также база данных с основными активностями и видами спорта. Эти базы данных пополняемы, то есть в любой момент времени пользователь может добавить тот или иной продукт или блюдо, вид спорта и т.д. Это позволяет сделать приложение сугубо индивидуальным, что является безусловным плюсом. Дальше все просто: каждый день вы добавляете продукты питания, потребляемые вами в течении дня, а также активности. Приложение показывает вам, сколько калорий вам нужно еще употребить. Тут все зависит от вашей цели. Если вы хотите похудеть, вам следует употреблять меньше, чем расходовать, если просто поддерживать форму, то количество затрачиваемых и потребляемых калорий должно быть одинаково, ну и соответственно, если массу наоборот нужно набрать нужно употреблять больше калорий. В этом приложение вам и поможет. Вы будете видеть, сколько вы съели, сколько израсходовали. Причем есть возможность отдельно отслеживать, сколько вы потратили за счет занятий, сколько просто за счет жизнедеятельности своего организма. А с помощью удобного календаря можно видеть данные прошедших дней.

Так же были рассмотрены аналоги данной программы: LifeSum, PumpUp, FatSecret . Проанализированы их плюсы и недостатки, и выбраны именно те качества, которые мы хотели видеть в своей программе. Это понятный пользователю интерфейс, возможность выбора продукта или активности из списка для его добавления в базу этого дня, добавление новых элементов в базы данных, просмотр данных предыдущих дней.

Были изучены возможности работы с базами данных в среде Swift.

Разработанное программное средство представляет собой законченный продукт, готовый к использованию. Однако при желании функционал программы можно расширить, добавив новые функции. Первое, что бы я хотела добавить это интегрирование с внутренним приложением HealthFit. Я начала изучение того, каким образом можно подключить мое приложение к HealthKit и как брать оттуда информацию о пройденных шагах. Я хочу, чтобы мое приложение в дальнейшем могло также отслеживать сколько калорий пользователь потратил за счет пройденных шагов. Также в приложение можно добавить индивидуальные программы похудения. Сделать так, чтобы приложение постоянно присылала уведомления пользователю с целью напоминания о том, что нужно вести дневник своего питания и активностей.

Таким образом создано максимально удобное средство для того, чтобы следить за своим образом жизни, которое всегда под рукой и не будет отнимать у вас много времени.

Список использованных источников

1. Developing iOS 10 Apps with Swift by Stanford. [Электронный ресурс] – Режим доступа: https://itunes.apple.com/ru/course/developing-ios-10-apps-with-swift/id1198467120?l=en - Стенфордский курс лекций для разработки приложений по IOS
2. Apple Developer. [Электронный ресурс] – Режим доступа: <https://developer.apple.com/>. – официальная документация по языку Swift
3. Сайт http://swiftbook.ru/ - русскоязычный источник информации о Swift (обработка json-данных)
4. Cайт https://videos.raywenderlich.com/courses/19-beginning-realm-on-ios/lessons/1- курс видео по использованию realm
5. Сайт https://github.com – источник библиотек для работы с базами данных и ответов на постоянно возникающие вопросы при трудностях с написанием кода
6. Сайт <http://stackoverflow.com> – источник ответов на вопросы

Приложение 1. Исходный код

1. Файл AppDelegate (вызывается при запуске программы, есть поясняющие комментарии)

//

// AppDelegate.swift

// FitApp

//

// Created by Kseniya Borokhovskaya on 4/22/17.

// Copyright © 2017 Kseniya Borokhovskaya. All rights reserved.

//

import UIKit

import RealmSwift

@UIApplicationMain

class AppDelegate: UIResponder, UIApplicationDelegate {

var window: UIWindow?

func application(\_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

let storyboard = UIStoryboard(name: "Main", bundle: nil)

// ifApplicationIsTheFirstTimeOnScreen

let defaults = UserDefaults.standard

if defaults.object(forKey: "isFirstTime") == nil {

PreloadData.getJSONActivitiesData() //get Activities Base in file "LoadJSONActivities

PreloadFoodData.getJSONAFoodData() //get Food Base in file "LoadJSONFood

defaults.set("No", forKey:"isFirstTime")

defaults.synchronize()

let viewController = storyboard.instantiateViewController(withIdentifier: "StartScreenViewController") // make the welcome screens

self.window?.rootViewController = viewController // appear first

self.window?.makeKeyAndVisible()

}

let tabViewController = storyboard.instantiateViewController(withIdentifier: "TabBarViewController") as? UITabBarController

tabViewController?.selectedIndex = 2

let dateFormatter = setDateFormatter() // in "Extensions" file

let currentDate = Date()

let today: String? = dateFormatter.string(from:currentDate as Date)

// check if we already make realm element for today

if let realm = try? Realm(),

let \_ = realm.object(ofType: DailyData.self, forPrimaryKey: today as AnyObject){ // if have do nothing

} else { // if not make it

let realm = try! Realm()

let todayData = DailyData()

try! realm.write

{

realm.add(todayData)

}

}

return true

}

func applicationWillResignActive(\_ application: UIApplication) {

// Sent when the application is about to move from active to inactive state. This can occur for certain types of temporary interruptions (such as an incoming phone call or SMS message) or when the user quits the application and it begins the transition to the background state.

// Use this method to pause ongoing tasks, disable timers, and invalidate graphics rendering callbacks. Games should use this method to pause the game.

}

func applicationDidEnterBackground(\_ application: UIApplication) {

// Use this method to release shared resources, save user data, invalidate timers, and store enough application state information to restore your application to its current state in case it is terminated later.

// If your application supports background execution, this method is called instead of applicationWillTerminate: when the user quits.

}

func applicationWillEnterForeground(\_ application: UIApplication) {

// Called as part of the transition from the background to the active state; here you can undo many of the changes made on entering the background.

}

func applicationDidBecomeActive(\_ application: UIApplication) {

// Restart any tasks that were paused (or not yet started) while the application was inactive. If the application was previously in the background, optionally refresh the user interface.

}

func applicationWillTerminate(\_ application: UIApplication) {

// Called when the application is about to terminate. Save data if appropriate. See also applicationDidEnterBackground:.

}

}

1. Файл StartScreenViewController (Контролирует экран, который пользователь видит при первой загрузке приложения)

//

// StartScreenViewController.swift

// LetFit

//

// Created by Kseniya Borokhovskaya on 4/27/17.

// Copyright © 2017 Kseniya Borokhovskaya. All rights reserved.

//

import UIKit

class StartScreenViewController: UIViewController {

@IBOutlet weak var StartImage: UIImageView!

override func viewDidLoad() {

super.viewDidLoad()

StartImage.image = UIImage(named: "Image") // set Image on Start Screen

}

}

1. Файл StartProfileTableViewController ( Контролирует работу экрана, загружаемого один раз при первой загрузке приложения для заполнения персональных данных)

//

// StartProfileTableViewController.swift

// LetFit

//

// Created by Kseniya Borokhovskaya on 4/28/17.

// Copyright © 2017 Kseniya Borokhovskaya. All rights reserved.

//

import UIKit

import RealmSwift

class StartProfileTableViewController: UITableViewController, UITextFieldDelegate{

// Outlets for text fields for entering personal information

@IBOutlet weak var name: UITextField!

@IBOutlet weak var surname: UITextField!

@IBOutlet weak var sex: UITextField!

@IBOutlet weak var age: UITextField!

@IBOutlet weak var weight: UITextField!

@IBOutlet weak var height: UITextField!

var user: Results<User>!

// controlls things that should be done before segueing to next screen

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

if segue.identifier == "toResults" { // our button "Continue" segue has identifier toResults, so if we click this button following actions will be done

let realm = try! Realm()

let user = User(name: name.text!,surname: surname.text!,sex: sex.text!, age: Int(age.text!)!, height: Int(height.text!)!, weight: Double(weight.text!)!) //set element of our user class

let todayData = realm.object(ofType: DailyData.self, forPrimaryKey: dateFormatter.string(from: (Date() as Date)) as AnyObject)

try! realm.write {

user.setBasalMetabolismAgain()

todayData?.leftFat = user.setFat()

todayData?.leftCarbs = user.setCarbs()

todayData?.leftProtein = user.setProtein()

realm.add(user)

}

// transport data to the next view

let storyboard = UIStoryboard(name: "Main", bundle: nil)

let viewController = storyboard.instantiateViewController(withIdentifier: "UserProfileTableViewController")

let profileController = viewController as! UserProfileTableViewController

let controller = segue.destination as! ResultsTableViewController

controller.detailUser = user

profileController.detailUser = user

}

}

// function which check if it time to perform segue (text fields are filled)

override func shouldPerformSegue(withIdentifier identifier: String, sender: Any!) -> Bool {

if identifier == "toResults" {

if (name.text?.isEmpty)! || (surname.text?.isEmpty)! || (sex.text?.isEmpty)! || (age.text?.isEmpty)! || (weight.text?.isEmpty)! || (name.text?.isEmpty)! || (height.text?.isEmpty)!{

let alert = UIAlertView()

alert.title = "No Information Entered"

alert.message = "Please enter your personal information"

alert.tintColor = UIColor.black

alert.addButton(withTitle: "Ok")

alert.show()

return false

}

else {

return true

}

}

return true

}

override func viewDidLoad() {

super.viewDidLoad()

height.delegate = self

}

func textFieldShouldReturn(\_ textField: UITextField) -> Bool {

textField.resignFirstResponder() // after clicking return button on keyboard it disappears

return true

}

}

1. Файл ResultsTableViewController ( Контролирует работу экрана, загружаемого один раз при первой загрузке приложения для показа результатов подсчета персональных данных(базовый метаболизм, белки, жиры, углеводы))

//

// ResultsTableViewController.swift

// LetFit

//

// Created by Kseniya Borokhovskaya on 4/28/17.

// Copyright © 2017 Kseniya Borokhovskaya. All rights reserved.

//

import UIKit

class ResultsTableViewController: UITableViewController {

@IBOutlet weak var basalMetabolism: UILabel!

@IBOutlet weak var carbs: UILabel!

@IBOutlet weak var protein: UILabel!

@IBOutlet weak var fat: UILabel!

var detailUser: User? {

didSet {

configureView()

}

}

// allows to transfer data from previous screen

func configureView() {

if let detailUser = detailUser {

if let basalMetabolism = basalMetabolism,

let carbs = carbs,

let protein = protein,

let fat = fat

{

basalMetabolism.text = String(Int(detailUser.basalMetabolism))

carbs.text = String(Int(detailUser.carbs))

protein.text = String(Int(detailUser.protein))

fat.text = String(Int(detailUser.fat))

}

}

}

override func viewDidLoad() {

super.viewDidLoad()

configureView()

}

}

1. Файл ViewController ( Контролирует работу главного экрана Home)

//

// ViewController.swift

// FitApp

//

// Created by Kseniya Borokhovskaya on 4/22/17.

// Copyright © 2017 Kseniya Borokhovskaya. All rights reserved.

//

//

import UIKit

import RealmSwift

class ViewController: UIViewController, UITableViewDelegate, UITableViewDataSource {

@IBOutlet weak var tableView: UITableView!

@IBOutlet weak var dateLabel: UILabel!

@IBOutlet weak var basalMetabolism: UILabel!

@IBOutlet weak var caloriesEaten: UILabel!

@IBOutlet weak var caloriesBurned: UILabel!

@IBOutlet weak var carbs: UILabel!

@IBOutlet weak var protein: UILabel!

@IBOutlet weak var fat: UILabel!

var user: User!

var aPopupContainer: PopupContainer?

var testCalendar = Calendar(identifier: .gregorian)

var currentDate: Date! = Date() {

didSet {

setDate()

}

}

// number of rows where daily information shown

var numberOfActivityRows = 0

var numberOfFoodRows = 0

var numberOfRowsAtSection: [Int] = [0, 0]

let dateFormatter = setDateFormatter()

@IBOutlet weak var CalendarButton: UIBarButtonItem!

override func viewDidLoad() {

super.viewDidLoad()

currentDate = Date()

// set image on calendar button

let button = UIButton.init(type: .custom)

button.setImage(UIImage(named: "CalendarSelectedState"), for: UIControlState.normal)

button.setImage(UIImage(named: "CalendarSelectedState"), for: UIControlState.highlighted)

button.addTarget(self, action: #selector(ViewController.showCalendar(\_:)), for: UIControlEvents.touchUpInside)

button.frame = CGRect(x: 0, y: 0, width: 53, height: 51)

let barButton = UIBarButtonItem(customView: button)

self.navigationItem.leftBarButtonItem = barButton

self.tableView.register(UITableViewCell.self, forCellReuseIdentifier: "HomeActivityCell")

// allows to work with tableView

tableView.delegate = self

tableView.dataSource = self

}

override func viewWillAppear(\_ animated: Bool) {

super.viewWillAppear(true)

updateData()

}

func numberOfSections(in tableView: UITableView) -> Int {

return 2

}

func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {

var rows: Int = 0

if section < numberOfRowsAtSection.count {

rows = numberOfRowsAtSection[section]

}

return rows

}

func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {

// load data to rows

let cell : UITableViewCell = self.tableView.dequeueReusableCell(withIdentifier: "HomeActivityCell")! as UITableViewCell

if let realm = try? Realm(),

let todayData = realm.object(ofType: DailyData.self, forPrimaryKey: dateFormatter.string(from: currentDate as Date) as AnyObject) {

try! realm.write {

if indexPath.section == 0

{

cell.textLabel?.text = todayData.activity.elements[indexPath.row].nameOfActivity

cell.detailTextLabel?.text = String(describing: todayData.activityCalories[indexPath.row].intValue)

} else {

cell.textLabel?.text = todayData.food.elements[indexPath.row].nameOfFood

cell.detailTextLabel?.text = String(describing: todayData.foodCalories[indexPath.row].intValue)

}

}

}

return cell

}

// set names of heads of sections

func tableView(\_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {

if section == 1{

return "Food"

} else {

return "Activity"

}

}

func tableView(\_ tableView: UITableView, commit editingStyle: UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {

// deleting data from table view and base

if editingStyle == .delete {

if let realm = try? Realm(),

let todayData = realm.object(ofType: DailyData.self, forPrimaryKey: dateFormatter.string(from: currentDate as Date) as AnyObject) {

try! todayData.realm!.write {

if indexPath.section == 0

{

let data = todayData.activity.elements[indexPath.row]

todayData.realm!.delete(data)

let caloriesData = todayData.activityCalories.elements[indexPath.row]

todayData.realm!.delete(caloriesData)

}

else {

let data = todayData.food.elements[indexPath.row]

todayData.realm!.delete(data)

let caloriesData = todayData.foodCalories.elements[indexPath.row]

todayData.realm!.delete(caloriesData)

}

}

numberOfFoodRows = todayData.food.endIndex

numberOfActivityRows = todayData.activity.endIndex

numberOfRowsAtSection = [numberOfActivityRows, numberOfFoodRows]

tableView.deleteRows(at: [indexPath], with: .fade)

updateData()

}

}

}

// allow to update data in table view and information view

func updateData(){

if let realm = try? Realm(),

let todayData = realm.object(ofType: DailyData.self, forPrimaryKey: dateFormatter.string(from: currentDate as Date) as AnyObject) {

user = realm.objects(User.self)[0]

basalMetabolism.text = String(Int(user.basalMetabolism))

caloriesEaten.text = String(todayData.eatenCalories)

caloriesBurned.text = String(todayData.burnedCalories)

carbs.text = String(todayData.leftCarbs)

fat.text = String(todayData.leftFat)

protein.text = String(todayData.leftProtein)

numberOfFoodRows = todayData.food.endIndex

numberOfActivityRows = todayData.activity.endIndex

numberOfRowsAtSection = [numberOfActivityRows, numberOfFoodRows]

} else{

let realm = try? Realm()

user = realm?.objects(User.self)[0]

basalMetabolism.text = String(Int(user.basalMetabolism))

caloriesEaten.text = "0"

caloriesBurned.text = "0"

carbs.text = String(user.carbs)

fat.text = String(user.fat)

protein.text = String(user.protein)

numberOfFoodRows = 0

numberOfActivityRows = 0

numberOfRowsAtSection = [numberOfActivityRows, numberOfFoodRows]

}

tableView.reloadData()

}

@IBAction func showCalendar(\_ sender: UIButton) {

let xibView = Bundle.main.loadNibNamed("CalendarPopUp", owner: nil, options: nil)?[0] as! CalendarPopUp

xibView.calendarDelegate = self

xibView.selected = currentDate

PopupContainer.generatePopupWithView(xibView).show()

}

func setDate() {

let month = testCalendar.dateComponents([.month], from: currentDate).month!

let weekday = testCalendar.component(.weekday, from: currentDate)

let monthName = DateFormatter().monthSymbols[(month-1) % 12]

let week = DateFormatter().shortWeekdaySymbols[weekday-1]

let day = testCalendar.component(.day, from: currentDate)

dateLabel.text = "\(week), " + monthName + " " + String(day)

updateData()

}

}

extension ViewController: CalendarPopUpDelegate {

func dateChaged(date: Date) {

currentDate = date

}

}

1. Файл UserProfileClass (Файл класса, описывающего пользователя, тут хранятся персональные данные пользователя)

//

// UserProfileClass.swift

// LetFit

//

// Created by Kseniya Borokhovskaya on 4/28/17.

// Copyright © 2017 Kseniya Borokhovskaya. All rights reserved.

//

import Foundation

import RealmSwift

class User: Object {

dynamic var id = NSUUID().uuidString

dynamic var name = ""

dynamic var surname = ""

dynamic var sex = ""

dynamic var age = 0

dynamic var height = 0

dynamic var weight = 0.0

dynamic var basalMetabolism = 0.0

dynamic var carbs = 0.0

dynamic var protein = 0.0

dynamic var fat = 0.0

// count and set Basal metabolism

func setBasalMetabolismAgain()

{

var BasalMetabolism = 1300.0

if self.sex == "male"{

let weightWithCoef = 13.7 \* self.weight

let heightWithCoef = Double(5 \* self.height)

BasalMetabolism = 66.5 + weightWithCoef + heightWithCoef - 6.8 \* Double(self.age)

}

else{

let weightWithCoef = 9.6 \* self.weight

let heightWithCoef = 1.8 \* Double(self.height)

BasalMetabolism = 665 + weightWithCoef + heightWithCoef - 4.7 \* Double(self.age)

}

self.basalMetabolism = BasalMetabolism

}

// count and set carbs, protein and fat

func setCarbs() -> Double {

self.carbs = (self.basalMetabolism \* 0.4) / 4

return self.carbs

}

func setProtein() -> Double {

self.protein = (self.basalMetabolism \* 0.35) / 4

return self.protein

}

func setFat() -> Double {

self.fat = (self.basalMetabolism \* 0.25) / 9

return self.fat

}

// initialiser

convenience init(name: String, surname: String,sex: String, age: Int, height: Int, weight: Double)

{

self.init()

self.name = name

self.surname = surname

self.sex = sex

self.age = age

self.height = height

self.weight = weight

}

override class func primaryKey() -> String?{

return "id"

}

}

1. Файл UserProfileTableViewController (Контролирует экран профайла)

//

// UserProfileTableViewController.swift

// LetFit

//

// Created by Kseniya Borokhovskaya on 4/29/17.

// Copyright © 2017 Kseniya Borokhovskaya. All rights reserved.

//

import UIKit

import RealmSwift

class UserProfileTableViewController: UITableViewController {

@IBOutlet weak var name: UITextField!

@IBOutlet weak var surname: UITextField!

@IBOutlet weak var sex: UITextField!

@IBOutlet weak var age: UITextField!

@IBOutlet weak var weight: UITextField!

@IBOutlet weak var height: UITextField!

@IBOutlet weak var basalMetabolism: UILabel!

@IBOutlet weak var carbs: UILabel!

@IBOutlet weak var protein: UILabel!

@IBOutlet weak var fat: UILabel!

@IBOutlet weak var EditButton: UIBarButtonItem!

var editTextFieldToggle: Bool = false

let dateFormatter = setDateFormatter()

@IBAction func Edit(\_ sender: UIButton) {

editTextFieldToggle = !editTextFieldToggle

if editTextFieldToggle == true {

textFieldActive()

EditButton.title = "Done"

} else {

textFieldDeactive()

EditButton.title = "Edit"

}

}

// allows to edit textFields

func textFieldActive()

{

name.isEnabled = true

surname.isEnabled = true

sex.isEnabled = true

age.isEnabled = true

weight.isEnabled = true

height.isEnabled = true

}

// stod editing

func textFieldDeactive()

{

name.isEnabled = false

surname.isEnabled = false

sex.isEnabled = false

age.isEnabled = false

weight.isEnabled = false

height.isEnabled = false

configureResultsView()

configureProfileView()

updateData()

}

var detailUser: User? {

didSet {

configureResultsView()

configureProfileView()

}

}

override func viewDidLoad() {

super.viewDidLoad()

configureProfileView()

configureResultsView()

updateData()

}

// load personal data

func configureResultsView() {

if let detailUser = detailUser {

if let basalMetabolism = basalMetabolism,

let carbs = carbs,

let protein = protein,

let fat = fat

{

basalMetabolism.text = String(detailUser.basalMetabolism)

carbs.text = String(detailUser.carbs)

protein.text = String(detailUser.protein)

fat.text = String(detailUser.fat)

}

}

}

func configureProfileView() {

if let detailUser = detailUser {

if let name = name,

let surname = surname,

let sex = sex,

let age = age,

let weight = weight,

let hight = height

{

name.text = detailUser.name

surname.text = detailUser.surname

sex.text = detailUser.sex

age.text = String(detailUser.age)

weight.text = String(detailUser.weight)

hight.text = String(detailUser.height)

}

}

}

// update data after editing

func updateData(){

if let realm = try? Realm() {

let user = realm.objects(User.self)[0]

basalMetabolism.text = String(user.basalMetabolism)

name.text = String(user.name)

surname.text = String(user.surname)

sex.text = String(user.sex)

age.text = String(user.age)

height.text = String(Int(user.height))

weight.text = String(Int(user.weight))

carbs.text = String(Int(user.carbs))

fat.text = String(Int(user.fat))

protein.text = String(Int(user.protein))

}

}

}

1. Файл RealmClassForStofingDailyData (Файл класса для хранения информации о каждодневном питании и движении )

//

// RealmClassForStofingDailyData.swift

// LetFit

//

// Created by Kseniya Borokhovskaya on 4/27/17.

// Copyright © 2017 Kseniya Borokhovskaya. All rights reserved.

//

import Foundation

import RealmSwift

let dateFormatter = setDateFormatter()

class RealmInt: Object {

dynamic var intValue = 0

}

class DailyData: Object {

// for saving added food

let food = List<Food>()

var foodCalories = List<RealmInt>()

var carbs = List<RealmInt>()

var fat = List<RealmInt>()

var protein = List<RealmInt>()

// for saving added activities

let activity = List<Activity>()

var activityCalories = List<RealmInt>()

// data for information view

dynamic var eatenCalories = 0

dynamic var burnedCalories = 0

dynamic var leftCarbs = 0.0

dynamic var leftFat = 0.0

dynamic var leftProtein = 0.0

dynamic var date = Date()

dynamic var DailyDataId = dateFormatter.string(from: Date() as Date)

override class func primaryKey() -> String?

{

return "DailyDataId"

}

}

1. Файл Try.json (JSON-файл с базой данных активностей)

{

"Activities" :

{

"1" : {

"nameOfActivity" : "Cycling 6 km/h",

"calories" : 300

},

"2" : {

"nameOfActivity" : "Cycling 8 km/h",

"calories" : 480

},

"3" : {

"nameOfActivity" : "Cycling 10 km/h",

"calories" : 600

},

"4" : {

"nameOfActivity" : "Cycling 16 km/h",

"calories" : 860

},

"5" : {

"nameOfActivity" : "Swimming",

"calories" : 476

},

"6" : {

"nameOfActivity" : "Jogging 6 km/h",

"calories" : 237

},

"7" : {

"nameOfActivity" : "Jogging 8 km/h",

"calories" : 340

},

"8" : {

"nameOfActivity" : "Jogging 10 km/h",

"calories" : 450

},

"9" : {

"nameOfActivity" : "Jogging 12 km/h",

"calories" : 500

},

"10" : {

"nameOfActivity" : "Playing football",

"calories" : 384

}

}

}

//json-stored database of activities

1. Файл Food.json (JSON-файл с базой данных продуктов)

{

"Food" :

{

"1" : {

"nameOfFood" : "Apple",

"calories" : 47,

"fat" : 0.4,

"carbohydrates" : 9.8,

"protein" : 0.4

},

"2" : {

"nameOfFood" : "Banana",

"calories" : 95,

"fat" : 0.2,

"carbohydrates" : 21.8,

"protein" : 1.5

},

"3" : {

"nameOfFood" : "Pinapple",

"calories" : 42,

"fat" : 0.3,

"carbohydrates" : 10.9,

"protein" : 0.4

},

"4" : {

"nameOfFood" : "Pasta",

"calories" : 116,

"fat" : 2.7,

"carbohydrates" : 14,

"protein" : 9.2

},

"5" : {

"nameOfFood" : "Pizza",

"calories" : 340,

"fat" : 9.4,

"carbohydrates" : 18.2,

"protein" : 13.7

},

"6" : {

"nameOfFood" : "Eggs",

"calories" : 157,

"fat" : 10.9,

"carbohydrates" : 0.7,

"protein" : 12.7

},

"7" : {

"nameOfFood" : "Bread",

"calories" : 323,

"fat" : 2.1,

"carbohydrates" : 55.5,

"protein" : 20.3

},

"8" : {

"nameOfFood" : "Cheese 0%",

"calories" : 71,

"fat" : 0.0,

"carbohydrates" : 1.3,

"protein" : 16.5

},

"9" : {

"nameOfFood" : "Cheese 4%",

"calories" : 104,

"fat" : 4.4,

"carbohydrates" : 1.4,

"protein" : 15.7

},

"10" : {

"nameOfFood" : "Cheese 9%",

"calories" : 159,

"fat" : 9.0,

"carbohydrates" : 2.0,

"protein" : 16.7

}

}

}

//json-stored database of food

1. Файл ActivitiesClass (Файл с классом активностей)

//

// ActivitiesClass.swift

// FitApp

//

// Created by Kseniya Borokhovskaya on 4/22/17.

// Copyright © 2017 Kseniya Borokhovskaya. All rights reserved.

//

import Foundation

import RealmSwift

class Activity: Object {

dynamic var id = NSUUID().uuidString

dynamic var nameOfActivity = ""

dynamic var calories = 0

override class func primaryKey() -> String?{

return "id"

}

}

1. Файл FoodClass (Файл с классом еды)

//

// FoodClass.swift

// LetFit

//

// Created by Kseniya Borokhovskaya on 4/27/17.

// Copyright © 2017 Kseniya Borokhovskaya. All rights reserved.

//

import Foundation

import RealmSwift

class Food: Object {

dynamic var FoodId = NSUUID().uuidString

dynamic var nameOfFood = ""

dynamic var calories = 0

dynamic var fat = 0.0

dynamic var protein = 0.0

dynamic var carbohydrates = 0.0

override class func primaryKey() -> String?

{

return "FoodId"

}

}

1. Файл LoadJSONActivities (Загружаем базу данных активностей, которые должны быть при загрузке приложения)

//

// LoadJSONActivities.swift

// FitApp

//

// Created by Kseniya Borokhovskaya on 4/23/17.

// Copyright © 2017 Kseniya Borokhovskaya. All rights reserved.

//

import Foundation

import RealmSwift

import SwiftyJSON

class PreloadData

{

static func getJSONActivitiesData()

{

let realm = try! Realm()

if let path = Bundle.main.path(forResource: "Try", ofType: "json") { //get connection to json-file with activities

do {

let data = try NSData(contentsOfFile: path, options: NSData.ReadingOptions.mappedIfSafe)

do {

let clearJSONActivities = JSON(data: data as Data, options: JSONSerialization.ReadingOptions.mutableContainers, error: nil)

let numberOfActivities = clearJSONActivities["Activities"].count

// fill realm base with Activities base

for i in 0..<numberOfActivities{

let activity = Activity()

activity.nameOfActivity = clearJSONActivities["Activities","\(i + 1)", "nameOfActivity"].stringValue

activity.calories = clearJSONActivities["Activities", "\(i + 1)", "calories"].intValue

try! realm.write

{

realm.add(activity)

}

}

}

} catch {}

}

}

}

1. Файл LoadJSONFood (Загружаем базу данных продуктов, которые должны быть при загрузке приложения)

//

// LoadJSONFood.swift

// LetFit

//

// Created by Kseniya Borokhovskaya on 4/27/17.

// Copyright © 2017 Kseniya Borokhovskaya. All rights reserved.

//

import Foundation

import RealmSwift

import SwiftyJSON

class PreloadFoodData

{

static func getJSONAFoodData()

{

let realm = try! Realm()

if let path = Bundle.main.path(forResource: "Food", ofType: "json") { //get connection to json-file with food

do {

let data = try NSData(contentsOfFile: path, options: NSData.ReadingOptions.mappedIfSafe)

do {

let clearJSONFood = JSON(data: data as Data, options: JSONSerialization.ReadingOptions.mutableContainers, error: nil)

let numberOfFood = clearJSONFood["Food"].count

// fill realm base with Food base

for i in 0..<numberOfFood{

let food = Food()

food.nameOfFood = clearJSONFood["Food","\(i + 1)", "nameOfFood"].stringValue

food.calories = clearJSONFood["Food", "\(i + 1)", "calories"].intValue

food.fat = clearJSONFood["Food", "\(i + 1)", "fat"].doubleValue

food.carbohydrates = clearJSONFood["Food", "\(i + 1)", "carbohydrates"].doubleValue

food.protein = clearJSONFood["Food", "\(i + 1)", "protein"].doubleValue

try! realm.write

{

realm.add(food)

}

}

}

} catch {}

}

}

}

1. Файл CreateActivityTableViewController (Контролирует добавление нового элемента в базу данных активностей)

//

// CreateActivityTableViewController.swift

// FitApp

//

// Created by Kseniya Borokhovskaya on 4/23/17.

// Copyright © 2017 Kseniya Borokhovskaya. All rights reserved.

//

import UIKit

import RealmSwift

class CreateActivityTableViewController: UITableViewController {

@IBOutlet weak var newActivity: UITextField!

@IBOutlet weak var addCalories: UITextField!

// add new activity to database of activities

@IBAction func createActivity(\_ sender: UIButton)

{

let realm = try! Realm()

let activity = Activity()

if let nameOfNewActivity: String = newActivity.text

{

activity.nameOfActivity = nameOfNewActivity

if let amountOfCalories: Int = Int(addCalories.text!)

{

activity.calories = amountOfCalories

try! realm.write

{

realm.add(activity)

}

}

}

navigationController!.popViewController(animated: true)

}

override func viewDidLoad() {

super.viewDidLoad()

}

}

1. Файл CreateMealTableViewController (Контролирует добавление нового элемента в базу данных продуктов)

//

// CreateMealTableViewController.swift

// LetFit

//

// Created by Kseniya Borokhovskaya on 4/27/17.

// Copyright © 2017 Kseniya Borokhovskaya. All rights reserved.

//

import UIKit

import RealmSwift

class CreateMealTableViewController: UITableViewController {

@IBOutlet weak var nameOfMeal: UITextField!

@IBOutlet weak var calories: UITextField!

@IBOutlet weak var carbs: UITextField!

@IBOutlet weak var fat: UITextField!

@IBOutlet weak var protein: UITextField!

// add new meal or product to database of food

@IBAction func Create(\_ sender: UIButton) {

let realm = try! Realm()

let food = Food()

if let nameOfNewMeal: String = nameOfMeal.text,

let amountOfCalories: Int = Int(calories.text!),

let amountOfFat: Double = Double(fat.text!),

let amountOfCarbs: Double = Double(carbs.text!),

let amountOfProtein: Double = Double(protein.text!)

{

food.nameOfFood = nameOfNewMeal

food.calories = amountOfCalories

food.carbohydrates = amountOfCarbs

food.fat = amountOfFat

food.protein = amountOfProtein

try! realm.write

{

realm.add(food)

}

}

navigationController!.popViewController(animated: true)

}

override func viewDidLoad() {

super.viewDidLoad()

}

}

1. Файл ActivityTableViewCell (Контролирует поведение ячейки Table View)

//

// ActivityTableViewCell.swift

// FitApp

//

// Created by Kseniya Borokhovskaya on 4/23/17.

// Copyright © 2017 Kseniya Borokhovskaya. All rights reserved.

//

import UIKit

class ActivityTableViewCell: UITableViewCell {

@IBOutlet weak var Name: UILabel!

@IBOutlet weak var Calories: UILabel!

var id: String?

override func awakeFromNib() {

super.awakeFromNib()

}

// add informarion in cell

func configureWithActivity(\_ activity: Activity) {

id = activity.id

Name!.text = activity.nameOfActivity

Calories!.text = "\(activity.calories)"

}

}

1. Файл FoodTableViewCell (Контролирует поведение ячейки Table View)

//

// FoodTableViewCell.swift

// LetFit

//

// Created by Kseniya Borokhovskaya on 4/27/17.

// Copyright © 2017 Kseniya Borokhovskaya. All rights reserved.

//

import UIKit

class FoodTableViewCell: UITableViewCell {

@IBOutlet weak var Name: UILabel!

@IBOutlet weak var Calories: UILabel!

var id: String?

override func awakeFromNib() {

super.awakeFromNib()

}

// add information in cell

func configureWithFood(\_ food: Food) {

id = food.FoodId

Name!.text = food.nameOfFood

Calories!.text = "\(food.calories)"

}

}

1. Файл AddActivityTableViewController (Контролирует показ базы данных активностей на экране)

//

// AddActivityTableViewController.swift

// FitApp

//

// Created by Kseniya Borokhovskaya on 4/22/17.

// Copyright © 2017 Kseniya Borokhovskaya. All rights reserved.

//

import UIKit

import RealmSwift

import SwiftyJSON

class AddActivityTableViewController: UITableViewController {

var activitiesFromBase: Results<Activity>!

override func viewDidLoad() {

super.viewDidLoad()

let realm = try! Realm()

activitiesFromBase = realm.objects(Activity.self)

}

override func viewWillAppear(\_ animated: Bool) {

super.viewWillAppear(animated)

tableView.reloadData()

}

// MARK: - Table view data source

override func numberOfSections(in tableView: UITableView) -> Int {

return 1

}

override func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {

return activitiesFromBase.count

}

// fill table view with information

override func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {

let activity = activitiesFromBase[indexPath.row]

let cell = tableView.dequeueReusableCell(withIdentifier: "ActivityCell") as! ActivityTableViewCell;

cell.configureWithActivity(activity)

return cell

}

override func tableView(\_ tableView: UITableView, canEditRowAt indexPath: IndexPath) -> Bool {

return true

}

// allows to delete elements

override func tableView(\_ tableView: UITableView, commit editingStyle: UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {

if editingStyle == .delete {

try! activitiesFromBase.realm!.write {

let activity = self.activitiesFromBase[indexPath.row]

self.activitiesFromBase.realm!.delete(activity)

}

tableView.deleteRows(at: [indexPath], with: .fade)

}

}

// MARK: - Segues

// transfer information to destination view

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

if segue.identifier == "AddActivity" {

if let indexPath = tableView.indexPathForSelectedRow {

let activity : Activity

activity = activitiesFromBase[indexPath.row]

let controller = segue.destination as! SelectedActivityTableViewController

controller.detailActivity = activity

}

}

}

}

1. Файл AddFoodTableViewController (Контролирует показ базы данных продуктов на экране)

//

// AddFoodTableViewController.swift

// LetFit

//

// Created by Kseniya Borokhovskaya on 4/27/17.

// Copyright © 2017 Kseniya Borokhovskaya. All rights reserved.

//

import UIKit

import RealmSwift

import SwiftyJSON

class AddFoodTableViewController: UITableViewController {

var foodFromBase: Results<Food>!

override func viewDidLoad() {

super.viewDidLoad()

let realm = try! Realm()

foodFromBase = realm.objects(Food.self)

}

override func viewWillAppear(\_ animated: Bool) {

super.viewWillAppear(animated)

tableView.reloadData()

}

// MARK: - Table view data source

override func numberOfSections(in tableView: UITableView) -> Int {

return 1

}

override func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {

return foodFromBase.count

}

override func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {

let food = foodFromBase[indexPath.row]

let cell = tableView.dequeueReusableCell(withIdentifier: "FoodCell") as! FoodTableViewCell

cell.configureWithFood(food)

return cell

}

override func tableView(\_ tableView: UITableView, canEditRowAt indexPath: IndexPath) -> Bool {

return true

}

// delete

override func tableView(\_ tableView: UITableView, commit editingStyle: UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {

if editingStyle == .delete {

try! foodFromBase.realm!.write {

let food = self.foodFromBase[indexPath.row]

self.foodFromBase.realm!.delete(food)

}

tableView.deleteRows(at: [indexPath], with: .fade)

}

}

// transfer information

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

if segue.identifier == "AddMeal" {

if let indexPath = tableView.indexPathForSelectedRow {

let food : Food

food = foodFromBase[indexPath.row]

let controller = segue.destination as! SelectedFoodTableViewController

controller.detailFood = food

}

}

}

}

1. Файл SelectedActivityTableViewController (Контролирует экран добавления активностей в дневную базу данных)

//

// SelectedActivityTableViewController.swift

// LetFit

//

// Created by Kseniya Borokhovskaya on 4/27/17.

// Copyright © 2017 Kseniya Borokhovskaya. All rights reserved.

//

import UIKit

import RealmSwift

class SelectedActivityTableViewController: UITableViewController, UITextFieldDelegate{

@IBOutlet weak var ActivityName: UILabel!

@IBOutlet weak var CaloriesPerHour: UILabel!

@IBOutlet weak var minutes: UITextField!

@IBOutlet weak var FinalCalories: UILabel!

var calories: RealmInt = RealmInt(value: 0)

let dateFormatter = setDateFormatter()

@IBAction func AddActivityToDailyBase(\_ sender: UIButton)

{

if let realm = try? Realm(),

let todayData = realm.object(ofType: DailyData.self, forPrimaryKey: dateFormatter.string(from: (Date() as Date)) as AnyObject) {

try! realm.write {

todayData.activity.append(detailActivity!)

todayData.activityCalories.append(calories)

todayData.burnedCalories = todayData.burnedCalories + calories.intValue

}

// shows window with text that says that activity added successfully

let alert = UIAlertView()

alert.title = "Added"

alert.message = "You can see it in Home-page"

alert.tintColor = UIColor.black

alert.addButton(withTitle: "Ok")

alert.show()

} else {

// shows window with text that says that addition failed

let alert = UIAlertView()

alert.title = "Failed"

alert.message = "Enter information properly"

alert.tintColor = UIColor.black

alert.addButton(withTitle: "Ok")

alert.show()

}

}

var detailActivity: Activity? {

didSet {

configureView()

}

}

func configureView() {

if let detailActivity = detailActivity {

if let ActivityName = ActivityName, let CaloriesPerHour = CaloriesPerHour {

ActivityName.text = detailActivity.nameOfActivity

CaloriesPerHour.text = String(detailActivity.calories)

title = detailActivity.nameOfActivity

}

}

}

override func viewDidLoad() {

super.viewDidLoad()

configureView()

minutes.delegate = self

}

func textFieldShouldReturn(\_ textField: UITextField) -> Bool {

textField.resignFirstResponder()

calories = RealmInt(value: Int(textField.text!)! \* Int(CaloriesPerHour.text!)! / 60)

FinalCalories.text = String(describing: calories.intValue)

return true

}

}

1. Файл SelectedFoodTableViewController (Контролирует экран добавления еды в дневную базу данных)

//

// SelectedFoodTableViewController.swift

// LetFit

//

// Created by Kseniya Borokhovskaya on 4/27/17.

// Copyright © 2017 Kseniya Borokhovskaya. All rights reserved.

//

import UIKit

import RealmSwift

class SelectedFoodTableViewController: UITableViewController, UITextFieldDelegate{

@IBOutlet weak var servingSize: UITextField!

@IBOutlet weak var numberOfServings: UITextField!

@IBOutlet weak var mealName: UILabel!

@IBOutlet weak var caloriesPer100g: UILabel!

@IBOutlet weak var finalCalories: UILabel!

@IBOutlet weak var finalCarbs: UILabel!

@IBOutlet weak var finalFat: UILabel!

@IBOutlet weak var finalProtein: UILabel!

var calories: RealmInt = RealmInt(value: 0)

var carbs: RealmInt = RealmInt(value: 0)

var fat: RealmInt = RealmInt(value: 0)

var protein: RealmInt = RealmInt(value: 0)

let dateFormatter = setDateFormatter()

@IBAction func AddFoodTodailyBase(\_ sender: UIButton) {

if let realm = try? Realm(),

let todayData = realm.object(ofType: DailyData.self, forPrimaryKey: dateFormatter.string(from: (Date() as Date)) as AnyObject) {

try! realm.write {

todayData.food.append(detailFood!)

todayData.foodCalories.append(calories)

todayData.carbs.append(carbs)

todayData.fat.append(fat)

todayData.protein.append(protein)

todayData.eatenCalories = todayData.eatenCalories + calories.intValue

todayData.leftFat = todayData.leftFat - Double(fat.intValue)

todayData.leftCarbs = todayData.leftCarbs - Double(carbs.intValue)

todayData.leftProtein = todayData.leftProtein - Double(protein.intValue)

}

// shows window with text that says that activity added successfully

let alert = UIAlertView()

alert.title = "Added"

alert.message = "You can see it in Home-page"

alert.tintColor = UIColor.black

alert.addButton(withTitle: "Ok")

alert.show()

} else {

// shows window with text that says that addition failed

let alert = UIAlertView()

alert.title = "Failed"

alert.message = "Enter information properly"

alert.tintColor = UIColor.black

alert.addButton(withTitle: "Ok")

alert.show()

}

}

var detailFood: Food? {

didSet {

configureView()

}

}

func configureView() {

if let detailFood = detailFood {

if let mealName = mealName, let caloriesPer100g = caloriesPer100g {

mealName.text = detailFood.nameOfFood

caloriesPer100g.text = String(detailFood.calories)

title = detailFood.nameOfFood

}

}

}

override func viewDidLoad() {

super.viewDidLoad()

configureView()

numberOfServings.delegate = self

}

func textFieldShouldReturn(\_ textField: UITextField) -> Bool {

textField.resignFirstResponder()

calories = RealmInt(value: Int(textField.text!)! \* Int(servingSize.text!)! \* Int(caloriesPer100g.text!)! / 100)

finalCalories.text = String(describing: calories.intValue)

carbs = RealmInt(value: Int(textField.text!)! \* Int(servingSize.text!)! \* Int((detailFood?.carbohydrates)!) / 100)

finalCarbs.text = String(describing: carbs.intValue)

fat = RealmInt(value: Int(textField.text!)! \* Int(servingSize.text!)! \* Int((detailFood?.fat)!) / 100)

finalFat.text = String(describing: fat.intValue)

protein = RealmInt(value: Int(textField.text!)! \* Int(servingSize.text!)! \* Int((detailFood?.protein)!) / 100)

finalProtein.text = String(describing: protein.intValue)

return true

}

}