Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых»

Аракелян С.М., Бутковский О.Я., Самышкин В.Д.

«ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО РАЗРАБОТКЕ iOS-ПРИЛОЖЕНИЙ»

Методические указания к лабораторному практикуму по дисциплине “Портативные вычислительные системы” для студентов IV курса по направлению 01.03.02 – прикладная математика и информатика

Владимир 2016

Составители: Аракелян Сергей Мартиросович, Бутковский Олег Ярославович, Самышкин Владислав Дмитриевич.

Методические указания для выполнения лабораторных работ по дисциплине «Лабораторный практикум по разработке iOS приложений» / Владим. гос. уни-т имени Александра Григорьевича и Николая Григорьевича Столетовых; Аракелян Сергей мартиросович, Бутковский Олег Ярославович, Самышкин Владислав Дмитриевич.
– Владимир: Изд-во ВлГУ, 2016.- 86с.

Предложены лабораторные работы по курсу «Лабораторный практикум по разработке iOS приложений» для студентов IV курса по направлению 01.03.02 – прикладная математика и информатика. Данное методическое пособие рассматривается как практическое средство для освоения основ разработки iOS-приложений в среде программирования Xcode на языке программирования Swift для мобильных устройств. Практикум содержит краткое описание среды программирования Xcode на языке программирования Swift для мобильных устройств и методические указания для выполнения 10-ти лабораторных работ по созданию основных элементов мобильных приложений.

Пособие выполнено в рамках государственного задания ВлГУ на 2014/13 на выполнение государственных работ в сфере научно-технической деятельности.

Рецензент – Давыдов Н.Н.-профессор, доктор технических наук, директор ИМПиБН.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В методических указаниях приведены лабораторные работы по курсу «Лабораторный практикум по разработке iOS приложений». Лабораторные работы сопровождаются краткой теорией, что делает более удобным их выполнение.

В лабораторном практикуме выполнение работ предполагается на устройствах MacBook Air и iPhone 5s и выше с установленной операционной системой OS X El Capitan в среде программирования Xcode 7 на языке программирования Swift 2 и с установленной операционной системой iOS 9.

ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

Работа в лаборатории требует от студента соответствующей подготовки. Для выполнения лабораторной работы, кроме рассмотренной теоретической части также необходимо использовать литературу, которую рекомендует преподаватель на лекции.

В отчет по лабораторной работе необходимо внести:

- номер и название работы;
- цель работы;
- теоретическую часть;
- текст программы, таблицы, расчетные формулы, графики и т.д.
- вывод.

При работе в лаборатории следует быть внимательным и выполнять правила по технике безопасности. Инструкция по технике безопасности находится в лаборатории.

Работа заканчивается составлением краткого заключения (вывода), в котором следует указать:

- в какой программной среде реализована лабораторная работа;
- краткий анализ результатов.

РАЗРАБОТКА ЛАБОРАТОРНОГО ПРАКТИКУМА

В лабораторный практикум входит десять лабораторных работ на следующие темы:

1) “Знакомство со средой программирования Xcode”.
2) “Запуск проекта Xcode на iOS-устройстве”.
3) “Создание универсального интерфейса приложения с помощью Autolayout”.
4) “Обработка событий для объектов интерфейса”.
5) “Создание iOS-приложения с использованием загруженных изображений”.
6) “Создание iOS-приложения с использованием жестов”.
7) “Создание iOS-приложения с использованием нескольких контроллеров”.
8) “Создание iOS-приложения с использованием контроллера табличного вида”.
9) “Создание iOS-приложения с использованием базы данных Core Data”.
10) “Создание простейшего iOS-калькулятора”.

При разработке приложений за основу взята программная платформа UIKit (User Unterface Kit). Она обеспечивает важнейшую инфраструктуру, необходимую для создания iOS-приложений и управления ими:

1. Архитектура окон и областей отображения (View), необходимая для управления пользовательским интерфейсом iOS-приложений.
2. Хранение различных событий, необходимых для реагирования на пользовательский ввод.
3. Модель приложения, необходимая для приведения в действие основного цикла и взаимодействия с системой[5].

Интерфейс разработанных приложений построен на основе контроллеров двух видов: View Controller и Table View Controller. Описание класса контроллеров приложений было реализовано в файлах с расширением “.swift”. В описании класса таких контроллеров вызов стандартных методов, в случае, если они были объявлены, происходит в строгой последовательности, показанной на рисунке 2.1, где:

init() – контроллер инициализируется в памяти;

loadView() – создается представление (View), которым управляет контроллер;

viewDidLoad() – вызывается после того, как представление (View) было загружено в память;

viewWillAppear() – вызывается перед появлением представления (View) на экране (в случае анимационных переходов между контроллерами, вызывается до начала анимации);

viewWillAppear() – вызывается после появления представления (View) на экране (в случае анимационных переходов между контроллерами, вызывается по завершении анимации);

viewWillDisappear() и viewDidDisappear() – методы, аналогичные двум предыдущим, только события происходят в обратной последовательности;

viewDidUnload() – вызывается после того, как представление (View) было выгружено из памяти.

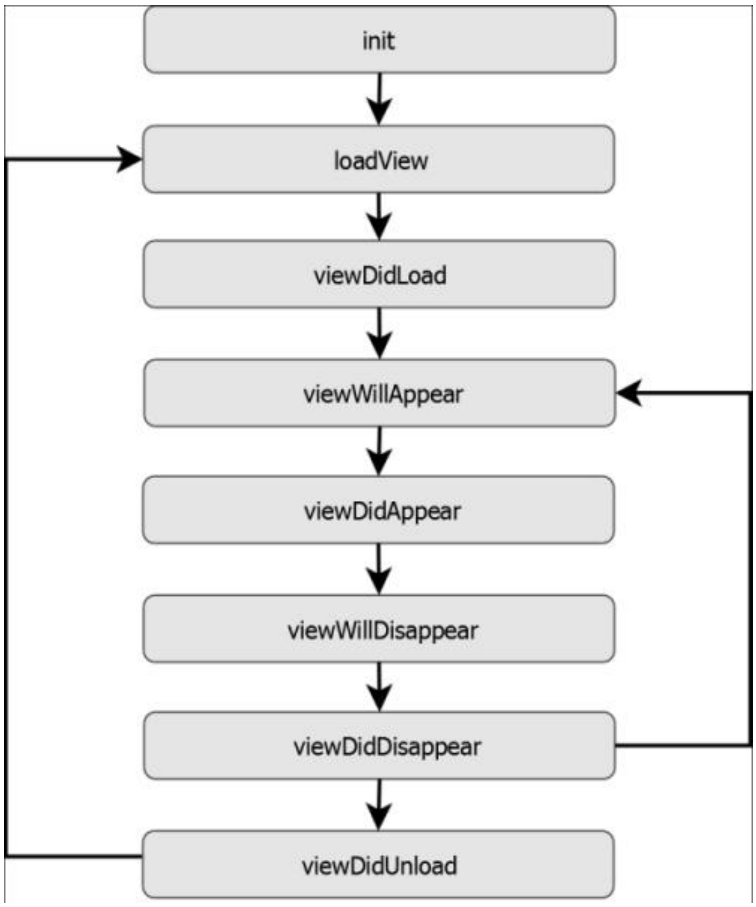


Рисунок 2.1 – Жизненный цикл контроллера View

В ходе выполнения работ лабораторного практикума используются методы viewDidLoad() и viewWillAppear(). В методе viewDidLoad() задаются дополнительные свойства для объектов интерфейса (лабораторные работы №5,6), а в методе viewWillAppear() реализуется загрузка элементов из базы данных и заполнение ими таблицы (лабораторная работа №9). Полные листинги приложений, разработанных для лабораторного практикума, приведены в приложении.

4.Методические указания к выполнению лабораторных работ

Методические указания для лабораторных работ содержат:

- тему работы;
- цель работы;
- ход работы.

Темы работ содержат их названия, передающие основную суть данной лабораторной работы.

В целях лабораторных работ указаны цели, которые студентам необходимо будет достичь по ходу выполнения. Также, в них указаны основные задания, на основе которых студентам будет необходимо разрабатывать приложения.

В ходе работ содержатся пошаговые указания, на основании которых студенты будут выполнять лабораторные работы. Лабораторные работы №1 и 2 содержат ознакомительный характер и ход работы для них отличается от хода работы для лабораторных работ №8.

На дальнейшем шаге хода работы содержатся инструкции по созданию связей между объектами интерфейса и кодом приложения. В нем указаны подробные действия, которые необходимо выполнить для того, чтобы связать объекты интерфейса с кодом приложения. Предпоследним шагом хода работы являются инструкции по написанию основного кода приложения. На этом шаге указаны действия, на основании которых необходимо реализовать в коде приложения работу объектов интерфейса, в соответствии с целью данной лабораторной работы.

На завершающем этапе хода работы необходимо собрать проект и запустить приложение в симуляторе iOS-устройства, либо на самом iOS-устройстве, если таковое имеется. В методических указаниях лабораторных работ на последнем шаге хода работы представлены скриншоты рабочего приложения, которое студентам было необходимо разработать.

1. ЛАБОРАТОРНАЯ РАБОТА №1 «ЗНАКОМСТВО СО СРЕДОЙ ПРОГРАММИРОВАНИЯ XCODE»

Цель работы:

Изучить среду Xcode. Научиться создавать проект iOS. Научиться запускать iOS Simulator.

Ход работы:

1.1 Установка среды программирования Xcode.

Для того, чтобы установить среду разработки Xcode, открываем приложение App Store. В поиске набираем Xcode и открываем страницу приложения (рис.1). В левом верхнем углу, под иконкой приложения, нажимаем кнопку “Загрузить”. Ждем окончания загрузки.

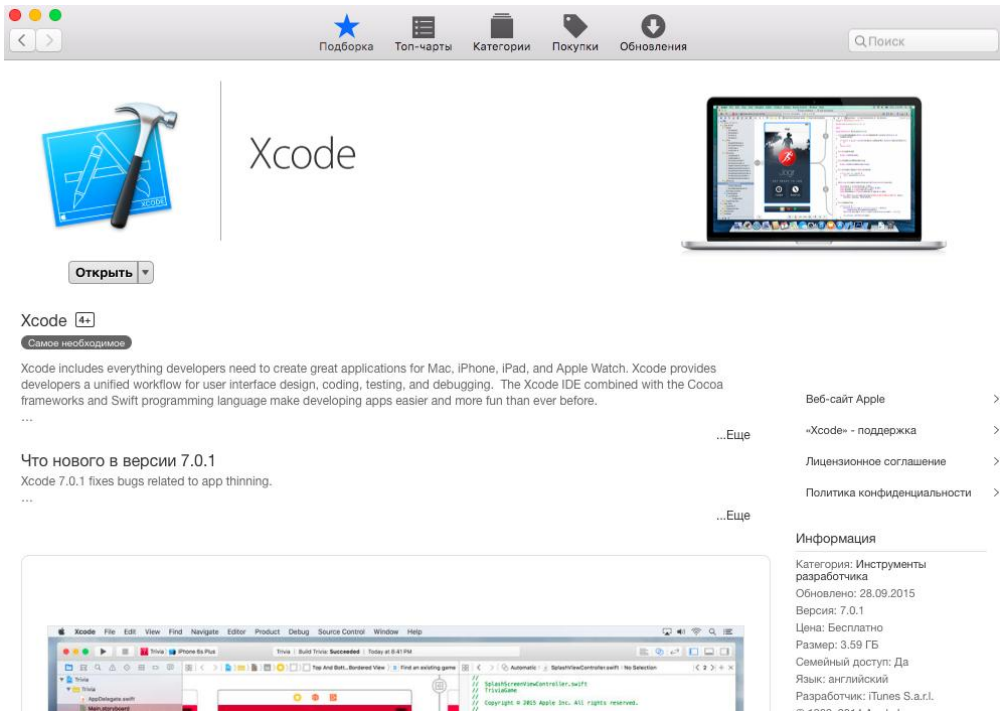


Рис.1. Страница приложения Xcode

1.2 Запуск среды программирования Xcode.

После завершения загрузки переходим в папку “Программы” на вашем компьютере Mac, либо нажимаем клавишу F4. Находим приложение Xcode и запускаем его (рис 2).

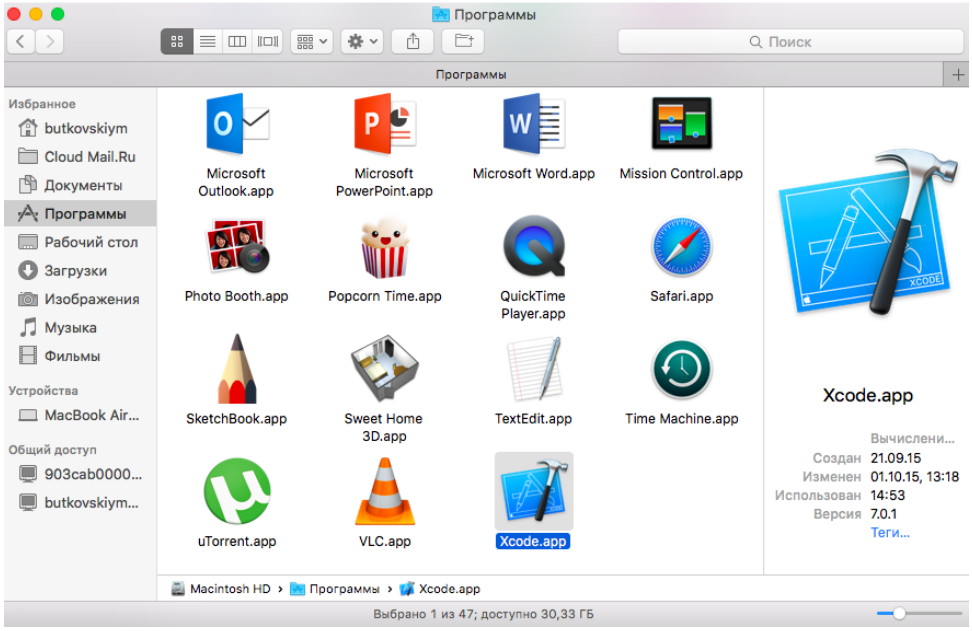


Рис. 2. Запуск программы

Появляется окно запуска (рис. 3), где мы можем:

- Быстро открыть недавние проекты (таблица в правом углу), либо открыть уже существующий проект;
- Открыть Playground (“песочница” для языка программирования Swift, где мы можем поэксперементировать с ним);
- Создать новый проект Xcode
- Просмотреть созданные проекты (начать работу с созданным ранее проектом)

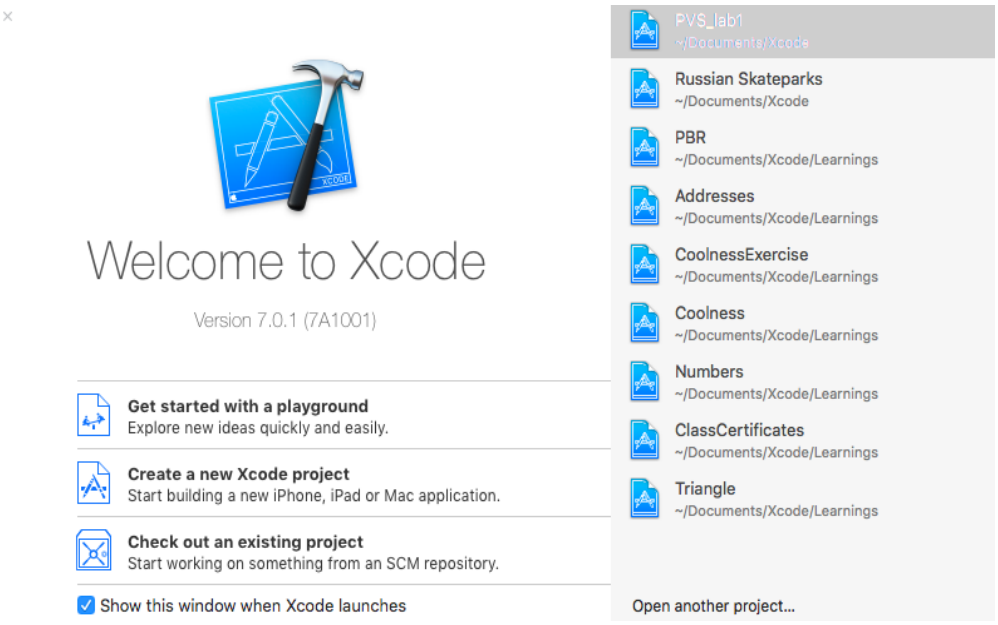


Рис.3. Окно запуска программы

Для того, чтобы это окно больше не открывалось при запуске, можно убрать галочку с параметра “Show this window when Xcode launches”.

1.3 Создание проекта в среде программирования Xcode.

Создаем новый проект, нажав на “Create a new Xcode project”.(рис.4)

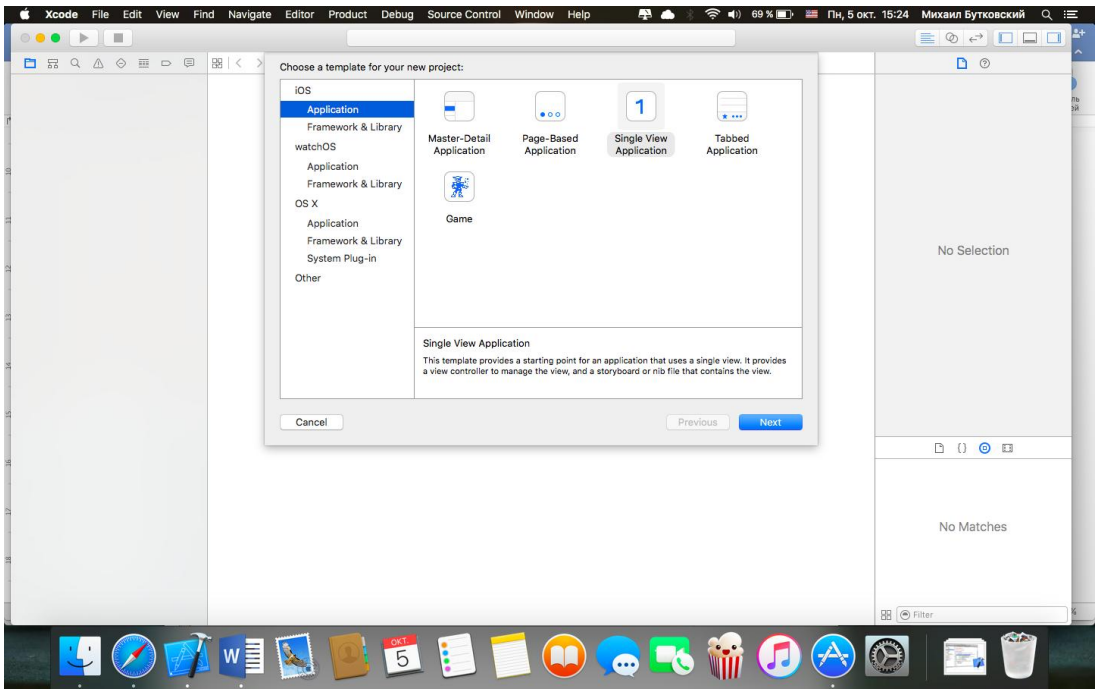


Рис.4. Создание нового проекта

Здесь мы можем выбрать шаблон, по которому будем создавать приложение для iOS (iPhone, iPad, iPod touch), watchOS (смарт-часы Apple Watch) или OS X (MacBook, MacBook Air, MacBook Pro, Mac mini, Mac Pro, iMac). Для каждой операционной системы имеется несколько шаблонов.

Выбираем iOS – Application – Single View Application (приложение с одним контроллером View Controller).

Нажимаем на кнопку “Next” и попадаем в окно выбора свойств нашего приложения (рис.5).

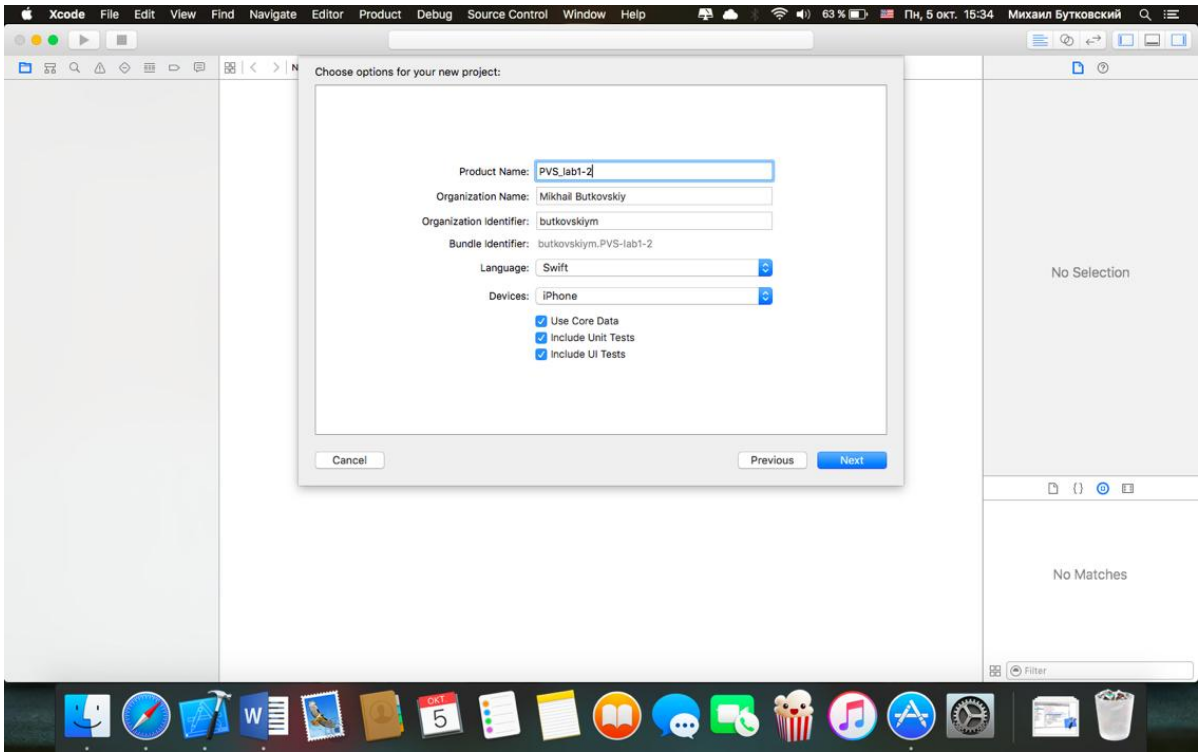


Рис.5. Окно выбора свойств

Здесь мы видим следующие параметры:

- Product Name – название приложения
- Organization Name – название компании разработчиков, либо имя индивидульного разработчика
- Organization Identifier – идентификатор организации, он же используется как идентификатор сборки (Bundle Identifier)
- Language – язык программирования
- Devices – устройство, на которое будет нацелен проект (проект может быть нацелен сразу на iPhone и iPad – параметр Universal)

- Use Core Data – использовать базу данных Core Data (встроенное в Xcode средство для работы с базами данных)
- Include Unit Tests – включить модульное тестирование (процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы)
- Include UI Tests – включить тестирование интерфейса пользователя

После того, как мы задали параметры для нашего приложения, нажимаем “Next”. Попадаем в окно выбора месторасположения нашего проекта, выбираем его и нажимаем “Create”.

1.4 Обзор интерфейса среды программирования Xcode и iOS Simulator.

Интерфейс среды программирования Xcode

Все параметры приложения заданы, шаблон выбран – мы попадаем в сам интерфейс Xcode (рис.6).

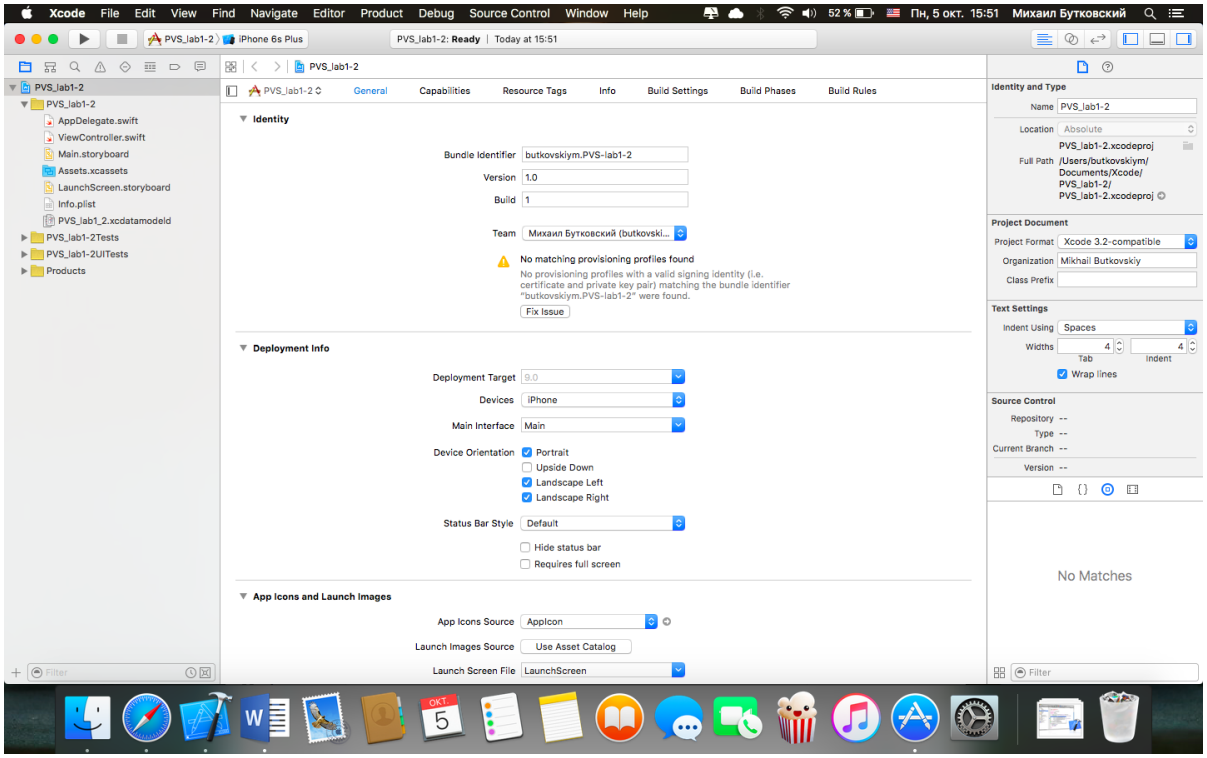


Рис. 6. Интерфейс Xcode

Главная часть интерфейса – это три области: (слева направо) область навигации, область редактирования и область утилит. В верхней части: (слева направо) кнопка компилирования и запуска приложения, выбор iOS симулятора и параметры схемы компилятора, строка состояния, кнопки редактора и кнопки вида.

iOS Simulator

iOS симулятор позволяет запускать написанные приложения на симуляторах различных устройств. Компания Apple настаивает на том, чтобы разработчики использовали инструменты только новейших операционных систем, в данном случае – iOS 9.

Для выбора устройства откройте список устройств в схеме компилятора.(рис.7). iOS Device в самой верхней части списка позволяет запустить приложение на вашем устройстве.

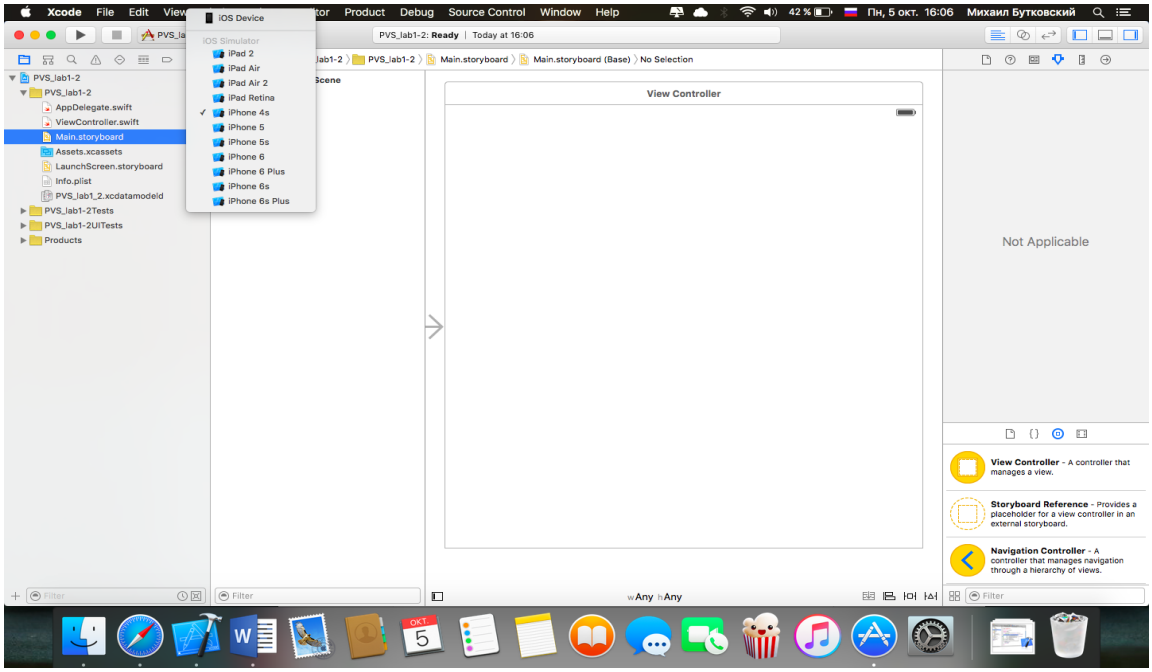


Рис. 7. Схема компилятора

1.5 Компиляция и запуск приложения на iOS Simulator.

Для запуска приложения в симуляторе iOS-устройств выполните команду cmd+R на клавиатуре, либо нажмите на кнопку сборки и запуска проекта (Build and then run the current scheme) в правом верхнем углу интерфейса (рис.8).

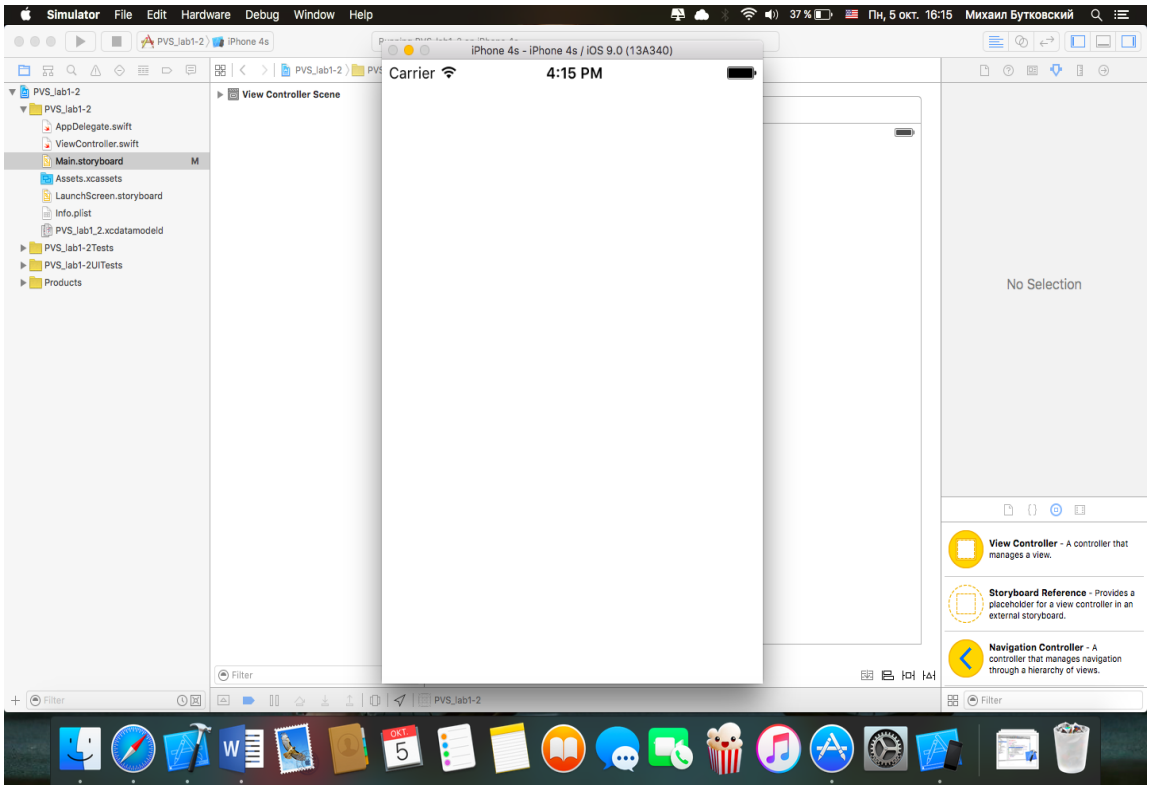


Рис. 8. Запуск приложения в симуляторе iOS

2. ЛАБОРАТОРНАЯ РАБОТА №2 «ЗАПУСК ПРОЕКТА XCODE НА IOS УСТРОЙСТВЕ »

Цель работы:

Научится запускать созданный в Xcode IDE проект на мобильном устройстве под управлением операционной системы iOS.

Ход работы:

2.1. Создание проекта в Xcode.

Запускаем приложение Xcode и создаем новый проект, по аналогии с предыдущей лабораторной работой (рис.9). Выбираем “Create a new Xcode project” – “iOS” – “Application” – “Single View Application”. Нажимаем кнопку “Next”, заполняем все оставшиеся поля, выбираем нужные свойства и указываем каталог нашего проекта. 1. Привязка своего Apple ID к Xcode.

Для этого выбираем в верхней строке “Xcode” – “Preferences” – “Accounts”. В левом нижнем углу списка Apple IDs нажимаем “+” – “Add Apple ID…”, вводим данные своего Apple ID и нажимаем кнопку “Add”. Мы видим, как в списке Apple IDs появился наш аккаунтом (рис.10). Тем самым мы привязали наш Apple ID к Xcode и можем запускать приложения на любых iOS устройствах, которые привязаны к этому же Apple ID.

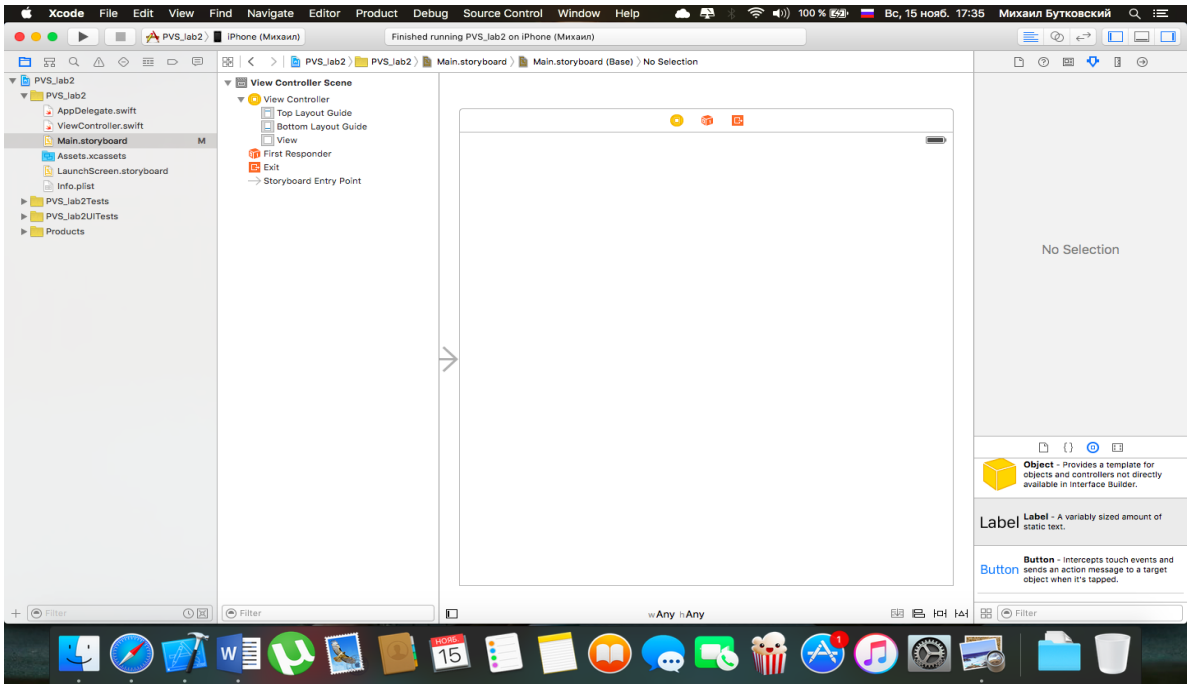


Рис. 9. Создание нового проекта

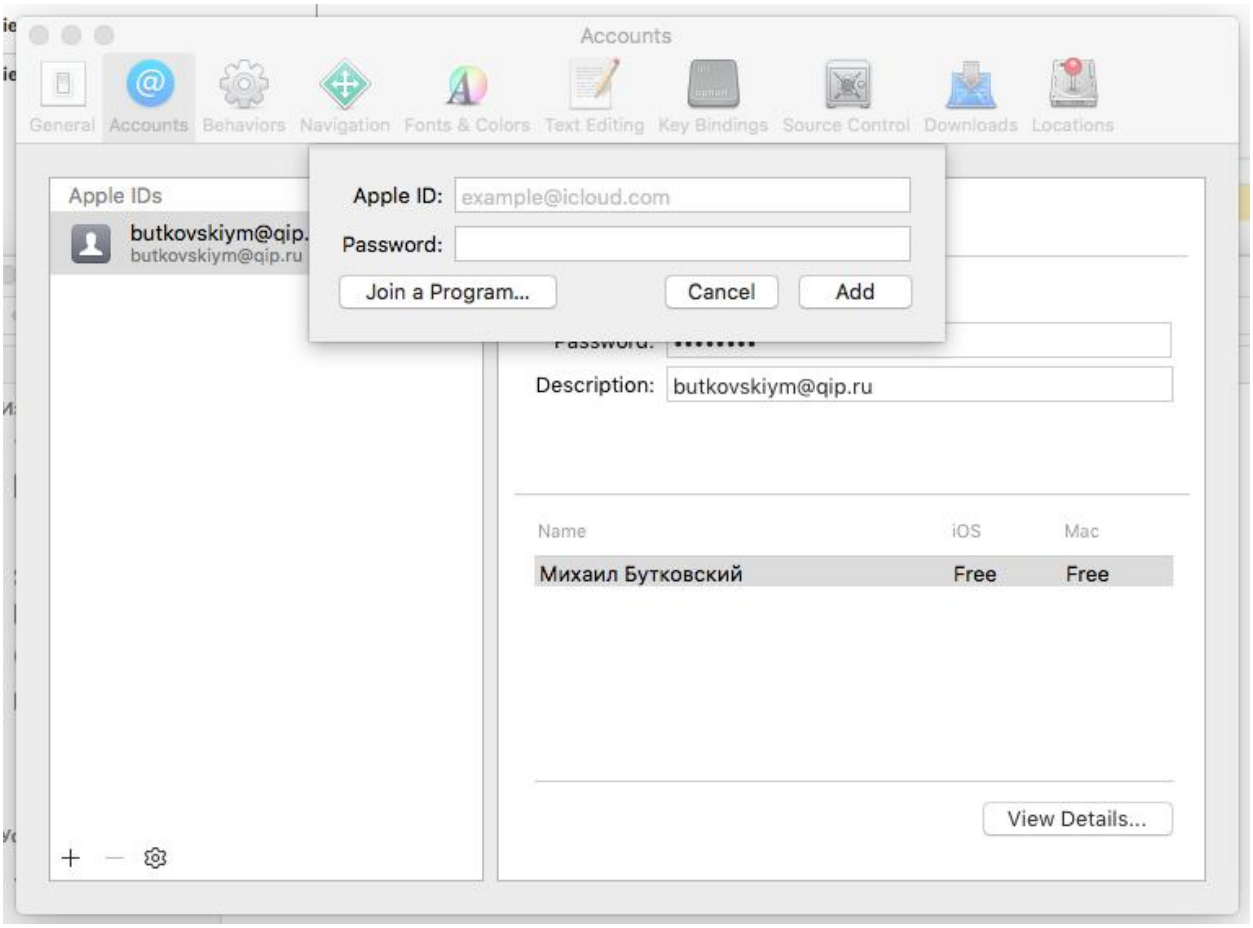


Рис. 10. Создание аккаунта

2.2 Добавление объекта Label на экран View Controller.

Теперь добавим статическое текстовое поле (объект UILabel) на экран нашего View контроллера и напишем в нем свою Фамилию и Имя.

Открываем файл Main.storyboard. В нижней части области утилит открываем библиотеку объектов (вкладка Show the object library), находим объект Label и перетаскиваем его в центр экрана контроллера.

Далее, выбираем наш View Controller и меняем его размер в соответствии с устройством (рис. 11), на котором вы будете запускать приложение. Для этого нажимаем на View Controller и, в верхней части области утилит, выбираем вкладку “Show the Attributes inspector”. Находим параметр Size и меняем его.

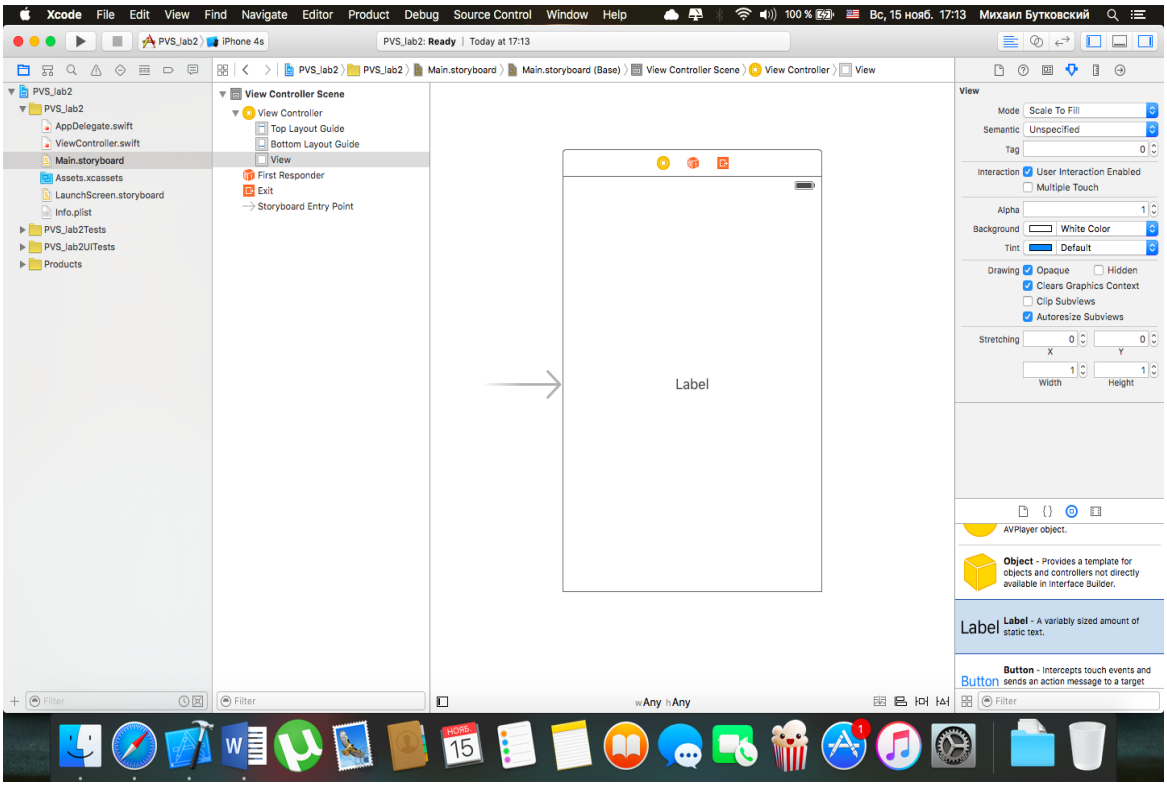


Рис. 11. Изменение размера View Controller

Растягиваем объект до краев экрана, дважды кликаем на него левой кнопкой мыши и пишем в нем свою Фамилию и Имя. Открываем свойства объекта Label и выравниваем текст по центру.

3. ЛАБОРАТОРНАЯ РАБОТА №3 «СОЗДАНИЕ УНИВЕРСАЛЬНОГО ИНТЕРФЕЙСА ПРИЛОЖЕНИЯ С ПОМОЩЬЮ AUTOLAYOUT »

Цель работы:

Создать универсальное приложение с помощью авто позиционирования (Autolayout), интерфейс которого будет одинаково хорошо выглядеть на всех iOS устройствах при любой ориентации экрана.

Ход работы:

3.1 Создание проекта Xcode.

Запускаем Xcode. Создаем проект, выбрав целевое устройство Universal в графе свойств Devices, а шаблон проекта: iOS – Application – Single View Application.

3.2 Добавление элементов интерфейса.

Добавим на экран View контроллера три объекта Label, поменяем цвет фона и зададим для них названия. Также, для одного из объектов, текст которого достаточно длинный, установим количество линий (Lines в свойствах объекта): 3, размер шрифта: 25 и минимальный размер шрифта (Autoshrink – Minimum Font Size): 5 в свойствах объекта. Теперь этот объект Label будет автоматически переносить текст на следующую строку в случае, если размер экрана iOS устройства будет меньшим, чем размер текста объекта Label. Также, размер шрифта текста объекта Label будет уменьшаться в диапазоне от 25 до 5, в зависимости от положения на экране какого-либо iOS устройства (рис.12).

3.3 Работа с Autolayout.

Для того, чтобы объекты на экране располагались правильно, в зависимости от устройства и ориентации экрана, необходимо добавить Constraints (связи). В правом нижнем углу находятся все инструменты для работы с Autolayout.

Для центрального объекта Label задаем оцентровку по горизонтали и вертикали. Также, добавим отступы слева и справа (leading и trailing) от краев экрана (рис. 13).

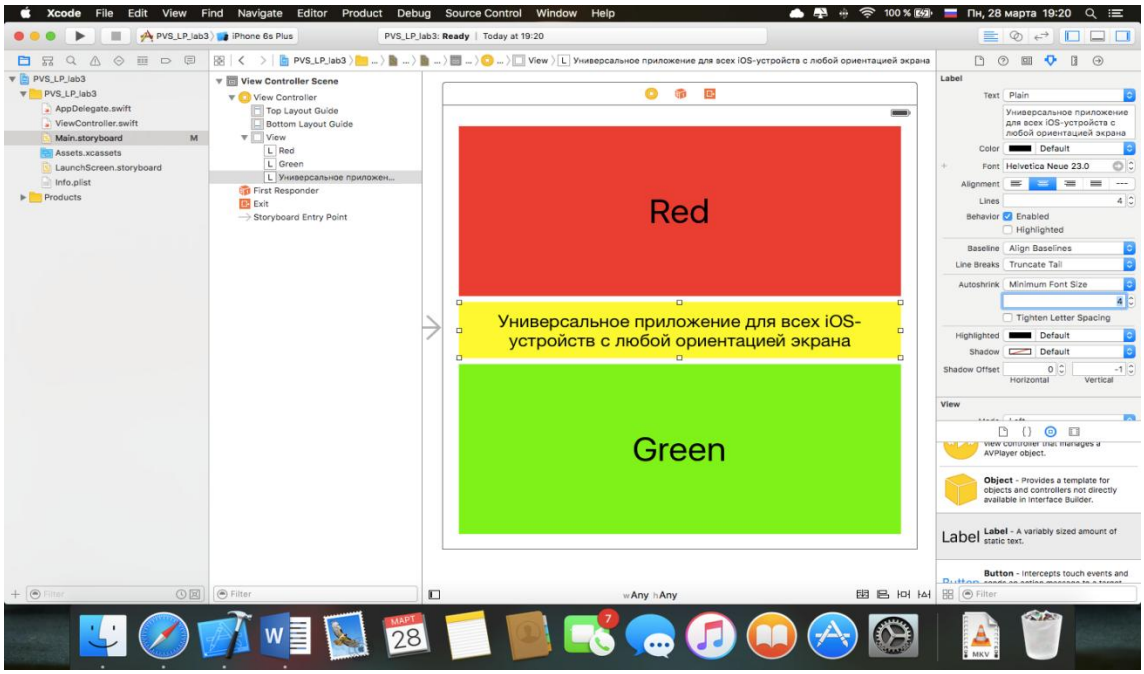


Рис. 12. Интерфейс устройства

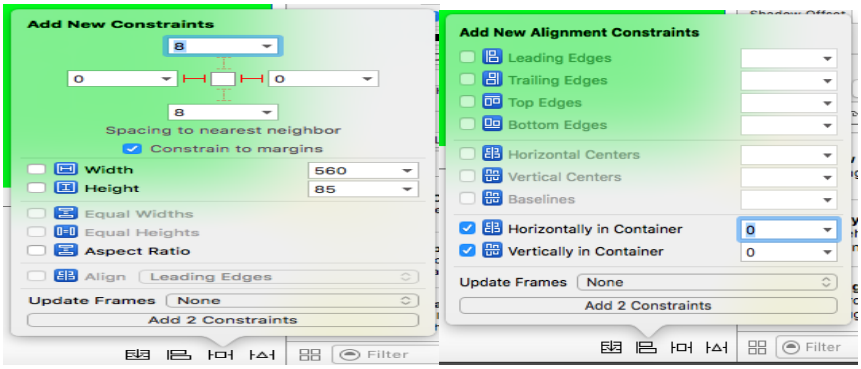


Рис.13. Добавление отступов

Для верхнего и нижнего объектов Label добавим отступы слева, справа, сверху и снизу (leading, trailing, top и bottom)(рис.14).

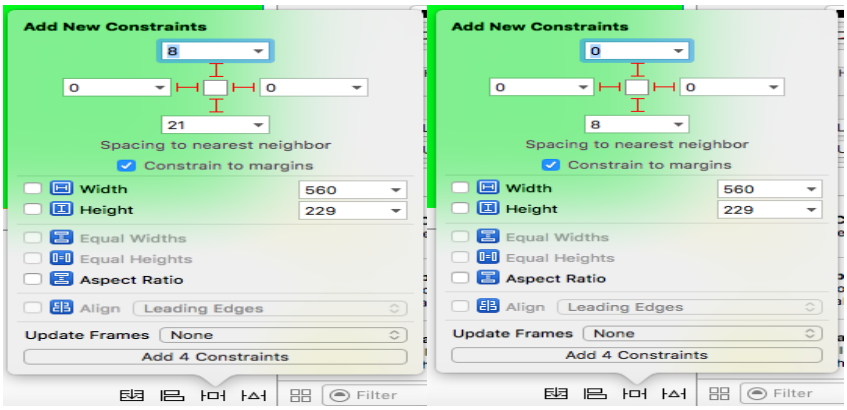


Рис.14. Отступы слева, справа, сверху и снизу

Список всех добавленных связей можно посмотреть в дереве объектов интерфейса, развернув ветку Constraints (рис.15).

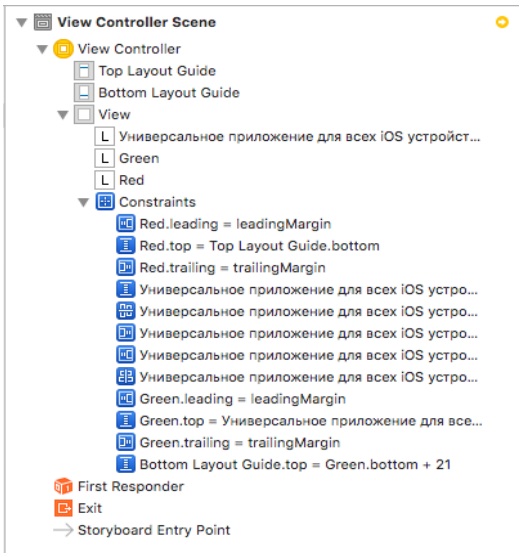


Рис.15. Дерево объектов интерфейса

Теперь, чтобы проверить, как будет выглядеть интерфейс нашего приложения на различных устройствах, открываем окно редактора-помощника (Show the Assistant Editor) и отображаем его снизу (Assistant Editors on Bottom).

В окне редактора-помощника выбираем Preview – Main.storyboard (Preview) и добавляем несколько различных устройств для проверки, нажав на + в левом нижнем углу редактора-помощника (рис.16).

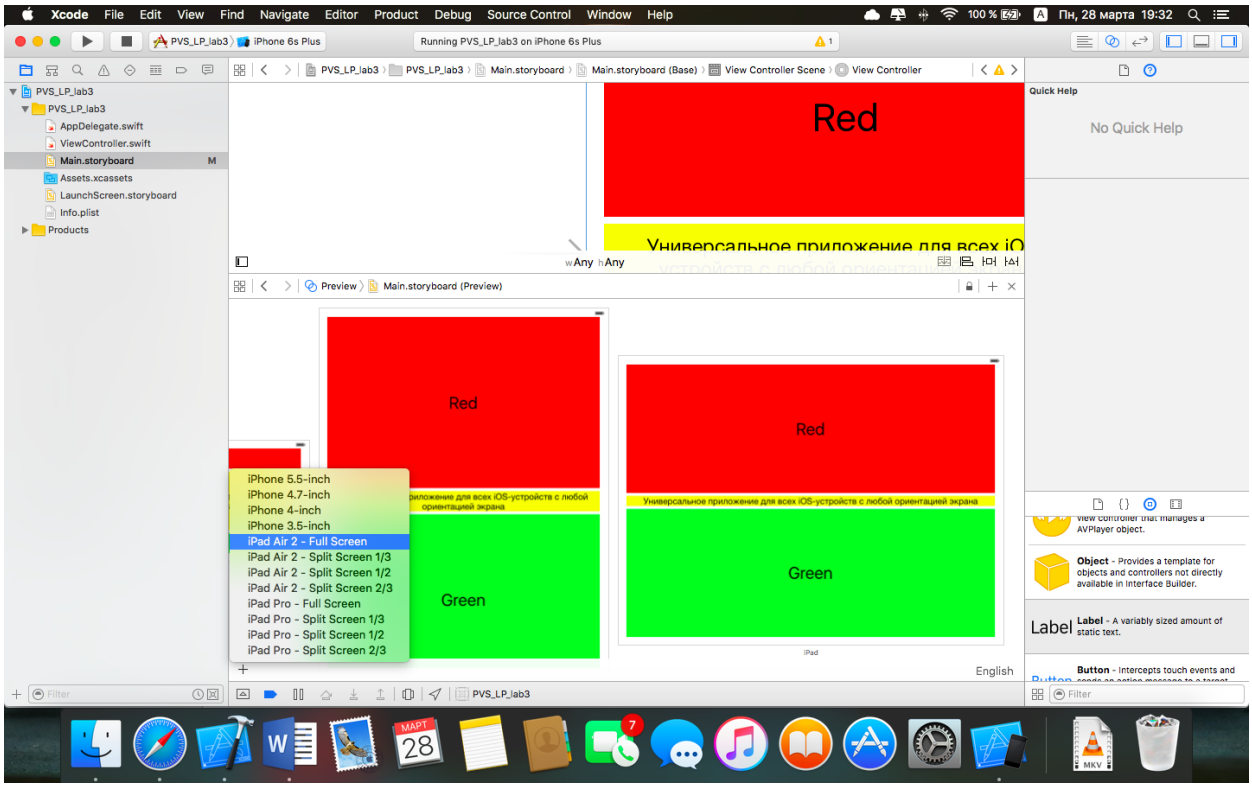


Рис.16. Окно редактора-помощника

3.4 Сборка и запуск проекта.

Соберем наш проект и запустим его на разных устройствах, поменяв у каждого ориентацию экрана.

Запуск на симуляторе iOS устройства iPhone 4s на рисунке 17.

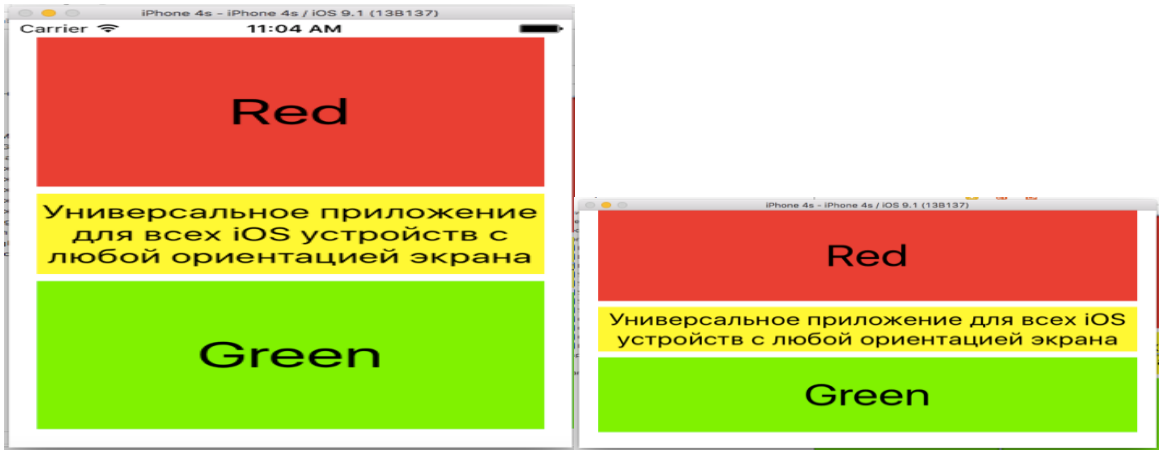


Рис.17. Запуск на симуляторе iOS устройства iPhone 4s

Запуск на симуляторе iOS устройства iPhone 6 на рисунке 18.

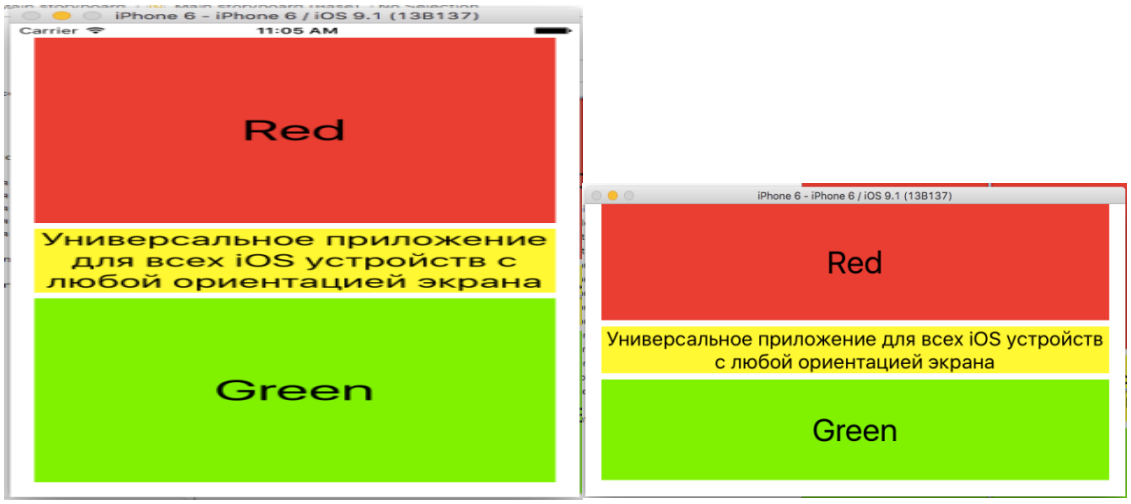


Рис.18. Запуск на симуляторе устройства iPhone 6

Запуск на симуляторе iOS устройства iPad Pro на рисунке 19.

Рис.19. Запуск на симуляторе устройства iPad Pro

4. ЛАБОРАТОРНАЯ РАБОТА №4 «ОБРАБОТКА СОБЫТИЙ ДЛЯ ОБЪЕКТОВ ИНТЕРФЕЙСА»

Цель работы:

Создать iOS приложение с использованием объектов “Button” и “Label” из библиотеки интерфейса UI (User Interface) для вывода текста “Привет” и “Пока” при нажатии кнопку 1, либо 2 – соответственно.

Ход работы:

4.1 Создание проекта Xcode.

Запускаем Xcode. Создаем проект iOS – Application – Single View Application, предварительно убрав галочку “Use Core Data”, т.к. база данных в этом приложении нам не понадобится.

4.2 Разработка интерфейса приложения.

Для этого в области навигации открываем файл Main.storyboard. Далее открываем библиотеку объектов (вкладка “Show the object library”) справа внизу, в области редактирования и перетаскиваем две кнопки “Button” и одно поле статического текста “Label” на наш ViewController в Main.storyboard.

Располагаем объекты на экране контроллера (рис.20) как нам удобно и изменяем их свойства: убираем текст “Label” и меняем название кнопок на “Кнопка 1” и “Кнопка 2”. Для этого выбираем нужный нам объект и в области редактирования открываем вкладку свойств объекта (Show the attributes inspector).

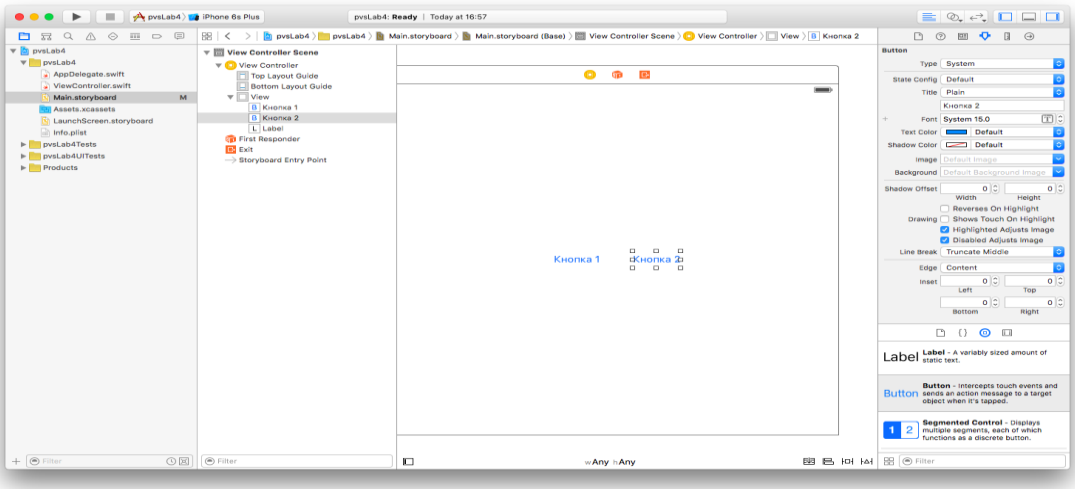


Рис.20. Объекты на экране контроллера

4.3 Создание связей между объектами интерфейса и кодом приложения.

Теперь нам необходимо объявить объекты интерфейса “Button” и “Label” в коде, т.е. – связать интерфейс программы с кодом приложения. В Xcode есть несколько способов это сделать. На мой взгляд, самый удобный из них – перетаскивание объекта в код.

На панели инструментов (рис.21), вверху справа, нажимаем на значок помощника (Show the Assistant editor). В области интерфейса экран разделится на две части, в каждой из которых мы можем отображать любые файлы из области навигации. В верхней части октроем файл “Main.storyboard”, а в нижней – файл с описанием класса нашего контроллера “ViewController.swift”. Выбираем объект, например Label, зажимаем клавишу ctrl и перетаскиваем его внутрь нашего класса. Отпускаем кнопку мыши и появляется окно со свойствами связи. В нем нам необходимо присвоить имя для объекта, например – “buttonText” и нажать кнопку “Connect”.

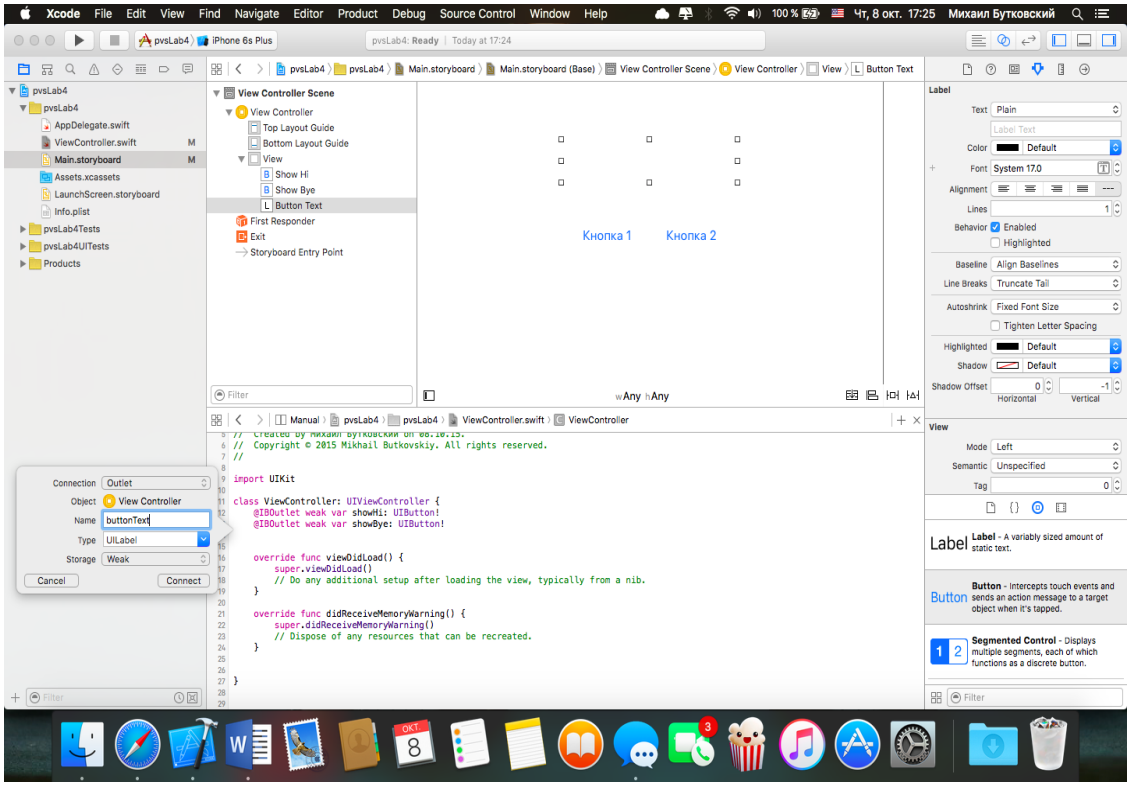


Рис. 21. Панель инструментов

Повторяем эти действия для оставшихся объектов. Должны появиться следующие строчки кода в описании класса:

```
@IBOutlet weak var showHi: UIButton!
@IBOutlet weak var showBye: UIButton!
@IBOutlet weak var buttonText: UILabel!
```

Это и есть наши связи объектов интерфейса с кодом приложения.

4.4 Обработка событий в коде приложения.

Объекты с интерфейсом связаны – осталось создать функции и описать действия, которые будут происходить, при нажатии на кнопки.

Для этого – выбираем первую кнопку в интерфейсе “Main.storyboard”, в области редактирования открываем вкладку с ее соединениями (Show the Connections inspector). В разделе событий (**Sent Events**) находим событие “Touch Up Inside” и перетаскиваем его в описание класса нашего контроллера (файл ViewController.swift). Задаем имя для функции, например – “hello” и нажимаем кнопку “Connect”. Повторяем эти действия для второй кнопки и описываем функции следующим образом:

```
@IBAction func hello(sender: AnyObject){
    buttonText.text = "Привет"
}

@IBAction func goodBye(sender: AnyObject) {
    buttonText.text = "Пока"
        }
```

В результате должно получиться как на рис.22.

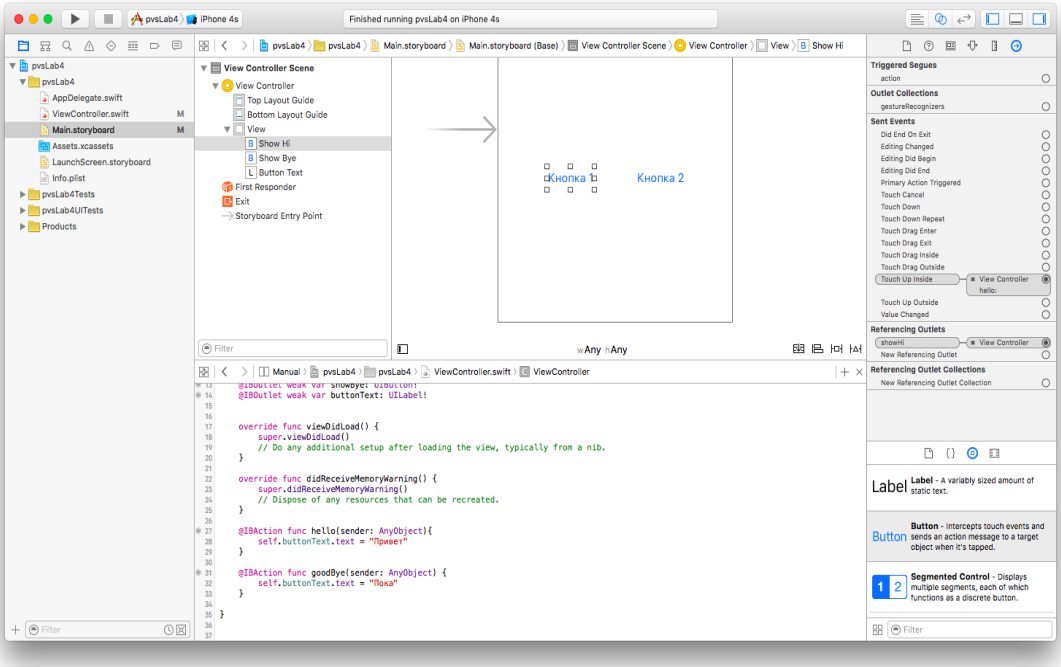
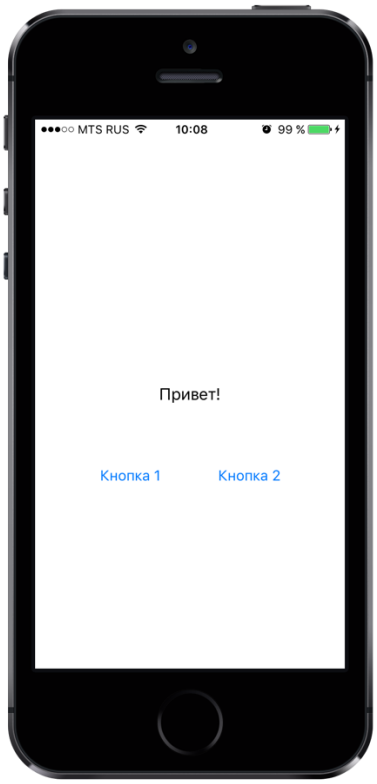


Рис. 22. Результат функции

Собираем наш проект и запускаем его, выполнив команду cmd+R.



5. ЛАБОРАТОРНАЯ РАБОТА №5 «СОЗДАНИЕ IOS-ПРИЛОЖЕНИЯ С ИСПОЛЬЗОВАНИЕМ ЗАГРУЖЕННЫХ ОБЪЕКТОВ МУЛЬТИМЕДИА»

Цель работы:

Создать iOS приложение с использованием объекта “Switch” для изменения фона контроллера. В положении “Вкл” – изображение “bg1.jpg”, в положении “Выкл” – изображение “bg2.jpg”. Изображения “bg1.jpg” и “bg2.jpg” ниже, слева направо соответственно (рис.23).

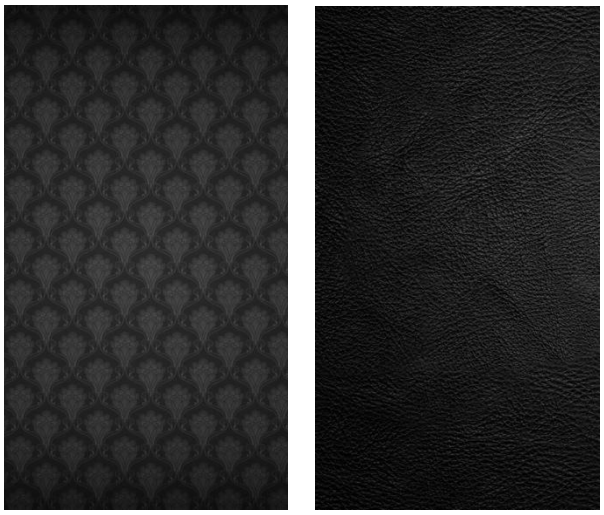


Рис.23. Изображения “bg1.jpg” и “bg2.jpg”

Ход работы:

5.1 Создание проекта Xcode.

Создаем проект Xcode, аналогично предыдущим лабораторным работам. Убираем галочку Use Core Data и выбираем iOS – Application – Single View Application.

5.2 Разработка интерфейса приложения.

Находим в библиотеке объектов (Object library) переключатель Switch и перетаскиваем его в окно нашего View контроллера в файле Main.storyboard. Располагаем его на экране как нам удобно. Также, добавляем объект Label, который будет индикатором переключателя Switch и располагаем его над переключателем.

Теперь необходимо связать объект Switch и Label с кодом нашей программы, аналогично лабораторной работе №4. Для этого открываем Assistant

Editor. В одном окне выбираем файл ViewController.swift, а в другом – Main.storyboard. С зажатой клавишей ctrl перетаскиваем объект Switch в описание класса ViewController. Отпускаем ЛКМ и задаем имя объекту: например, backgroundSwitch и нажимаем Connect. Должна появится следующая строчка кода (рис.24):

```
@IBOutlet weak var backgroundSwitch: UISwitch!
```

Повторяем эти действия для объекта Label:

```
@IBOutlet weak var switchIndicator: UILabel!
```

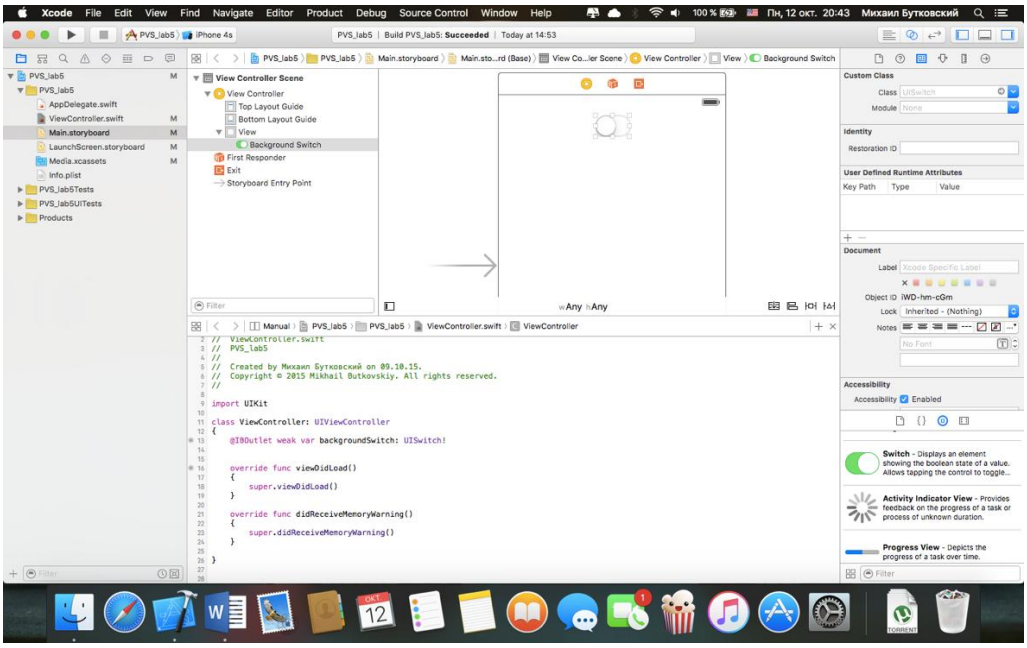


Рис.24. Результат кода

5.3 Добавление изображений в проект Xcode.

Для того, чтобы наше приложение использовало изображения, необходимо добавить их в специальное хранилище в нашем проекте. Для этого заранее изменяем разрешение изображений под разрешение iOS устройства, для которого вы создаете приложение (в данном случае – iPhone 5s, 640x1136 пикс.). Далее, в окне навигации открываем файл Assets.xcassets и добавляем набор изображений. Для этого нажимаем “+” (Add a group or image set) внизу списка и нажимаем “New Image Set”. Появляется окно с разными масштабами, под разные устройства (рис.25). Нам необходимо перетащить наше изображение в окошко 2x. Меняем название набора изображений с “Image” на “bg1”. Аналогично создаем набор изображений для “bg2”.

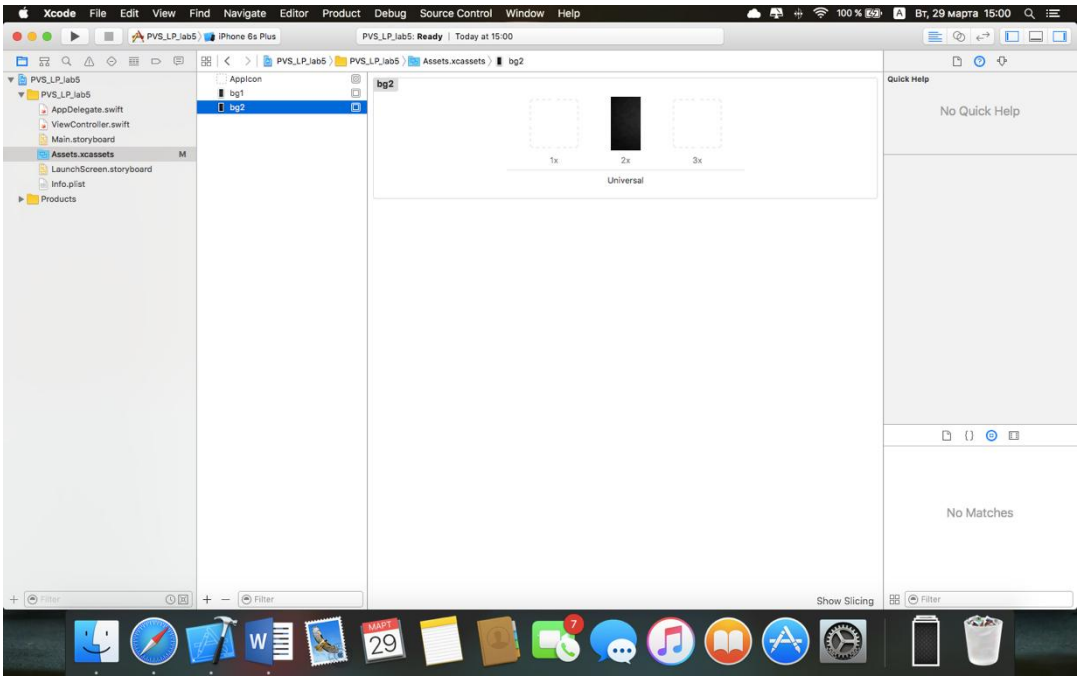


Рис.25. Окно с масштабами под разные устройства

5.4 Описание событий объекта “Switch” в коде, сборка и запуск приложения.

Осталось описать события, которые будут происходить при изменении положения выключателя. С зажатой клавишей ctrl добавляем соединение для объекта “Switch” чуть ниже, чем предыдущая связь. Выбираем тип соединения Action и задаем имя, например: “backgroundSwitchTapped” и нажимаем Connect. У нас в коде появилась функция, в которой можно описывать события для объекта “Switch”.

В теле функции пишем следующее условие:

```
@IBAction func backgroundSwitchTapped(sender: AnyObject)
  {
    if backgroundSwitch.on
    {
      switchIndicator.text = "Background image: bg1.jpg"
      view.backgroundColor = UIColor(patternImage: UIImage(named: "bg1")!)
    }
    else
    {
      switchIndicator.text = "Background image: bg2.jpg"
      view.backgroundColor = UIColor(patternImage: UIImage(named: "bg2")!)
```

```
    }
}
```

Добавляем в функцию ViewDidLoad начальное положение объекта Switch и начальный статус OFF индикатора Label.

```
override func viewDidLoad()
{
    super.viewDidLoad()

    switchIndicator.textColor = UIColor.whiteColor()
    switchIndicator.text = "Background image: bg2.jpg"
    view.backgroundColor = UIColor(patternImage: UIImage(named: "bg2")!)
}
```

5.5 Сборка проекта и запуск приложения.

Выполняем команду cmd+R и проверяем работу приложения.

6. ЛАБОРАТОРНАЯ РАБОТА №6 «СОЗДАНИЕ IOS ПРИЛОЖЕНИЯ С ИСПОЛЬЗОВАНИЕМ ЖЕСТОВ»

Цель работы:

Научиться обрабатывать события, которые будут происходить при использовании различных жестов.

Создать объект UILabel, который будет отображать используемый пользователем жест и менять цвет своего фона, в зависимости от используемого жеста. Реализовать работу следующих жестов:

1. Вращение (цвет фона: синий)
2. Масштабирование (цвет фона: красный)
3. Касание (цвет фона: зеленый)
4. Долгое нажатие (цвет фона: оранжевый)
5. Смахивание (цвет фона: серый)

Ход работы:

6.1 Создание проекта Xcode.

Создаем проект Xcode, аналогично предыдущим лабораторным работам. Убираем галочку Use Core Data и выбираем iOS – Application – Single View Application.

6.2 Добавление элементов интерфейса.

Открываем файл Main.storyboard и на экран контроллера View добавляем объект Label. Располагаем его посередине экрана и растягиваем (рис.26).

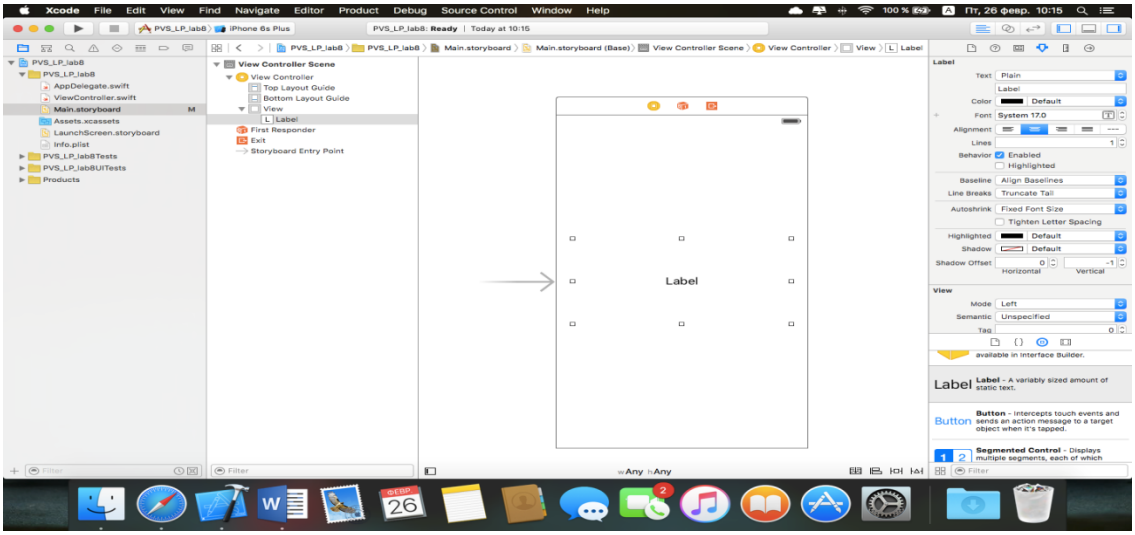


Рис.26. Добавление объекта Label

Далее, находим объекты: Tap Gesture Recognizer, Long Press Gesture Recognizer, Pinch Gesture Recognizer, Rotation Gesture Recognizer и Swipe Gesture Recognizer. Перетаскиваем их в область объекта Label (рис.27).

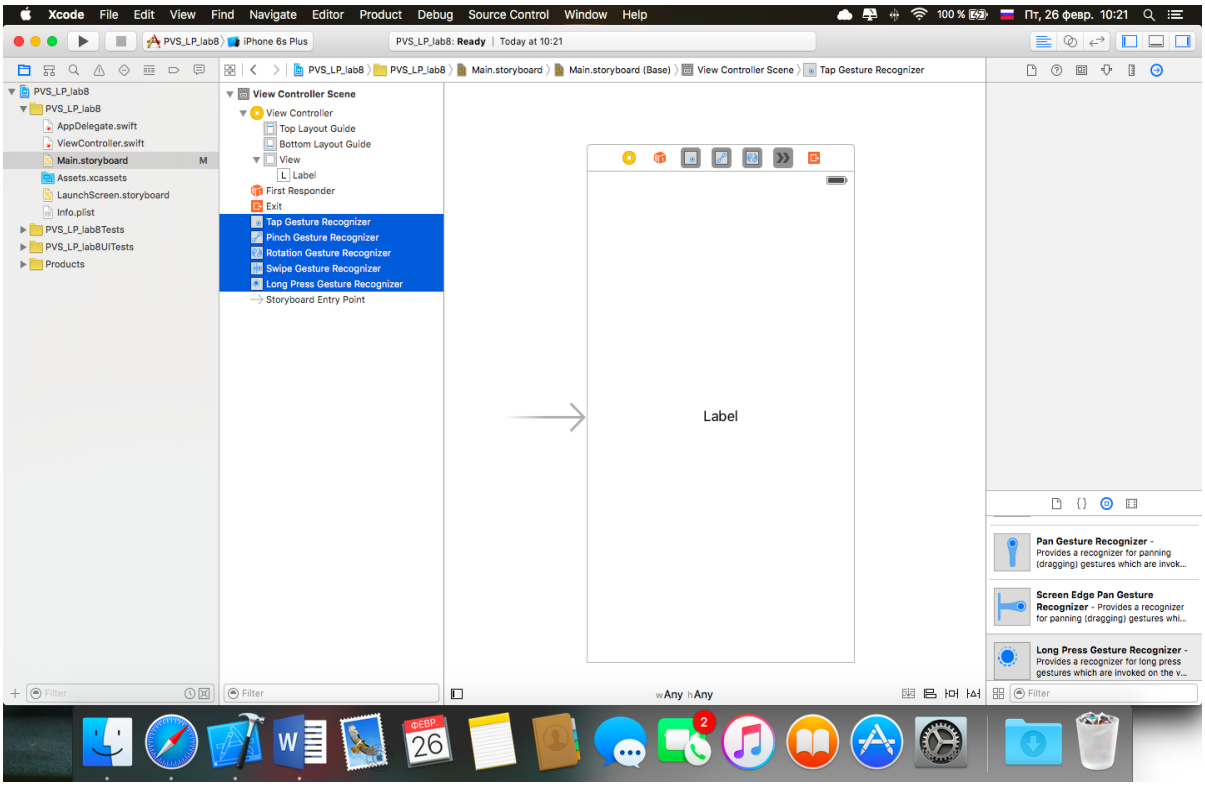


Рис.27. Область объекта Label

6.3 Создание связей объектов интерфейса с кодом приложения.

В описание класса контроллера View для объекта Label добавляем связь типа Outlet:

```
@IBOutlet weak var gestureIndicator: UILabel!
```

Для объектов Gesture Recognizer добавляем пять связей типа Action:

```
@IBAction func tap(sender: AnyObject)
{

    }
@IBAction func pinch(sender: AnyObject)
{

  }
```

```
@IBAction func rotation(sender: AnyObject)
{

}

@IBAction func swipe(sender: AnyObject)
{

    }

    @IBAction func longPress(sender: AnyObject)
    {

    }
```

6.4 Описание событий в коде приложения

Для того, чтобы использование жестов в области объекта Label стало возможным, присвоим параметру userInteractionEnabled (взаимодействие с пользователем включено) значение true в методе ViewDidLoad():

```
gestureIndicator.userInteractionEnabled = true
```

В этом же методе, задаем выравнивание текста, количество строк текста, текст и цвет фона:

```
gestureIndicator.textAlignment = NSTextAlignment.Center
gestureIndicator.numberOfLines = 2
gestureIndicator.text = "Используйте жесты в этой области"
gestureIndicator.backgroundColor = UIColor.yellowColor()
```

Далее, опишем события для всех пяти жестов в созданных ранее функциях. Для жеста касания:

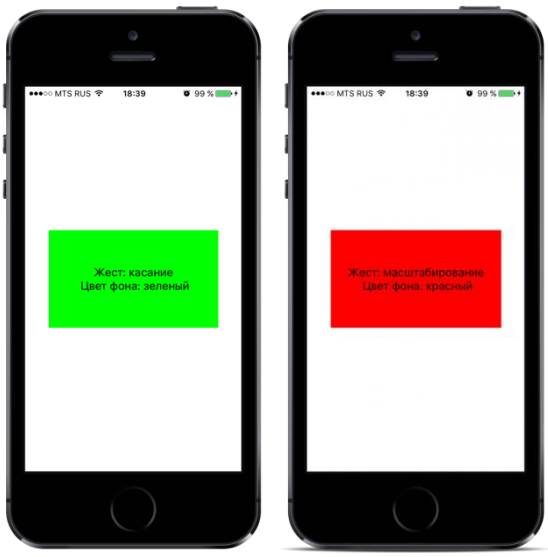
```
    @IBAction func tap(sender: AnyObject)
    {
        gestureIndicator.text = "Жест: касание\n Цвет фона: зеленый"
        gestureIndicator.backgroundColor = UIColor.greenColor()
    }
```

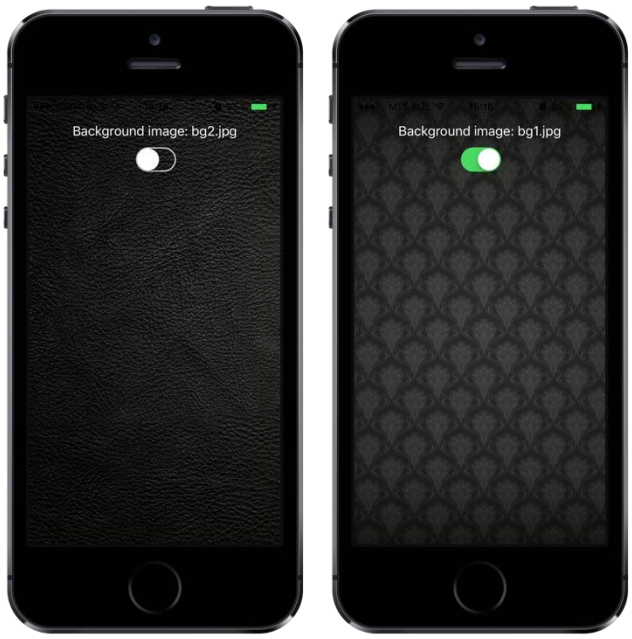
Для жеста масштабирования:

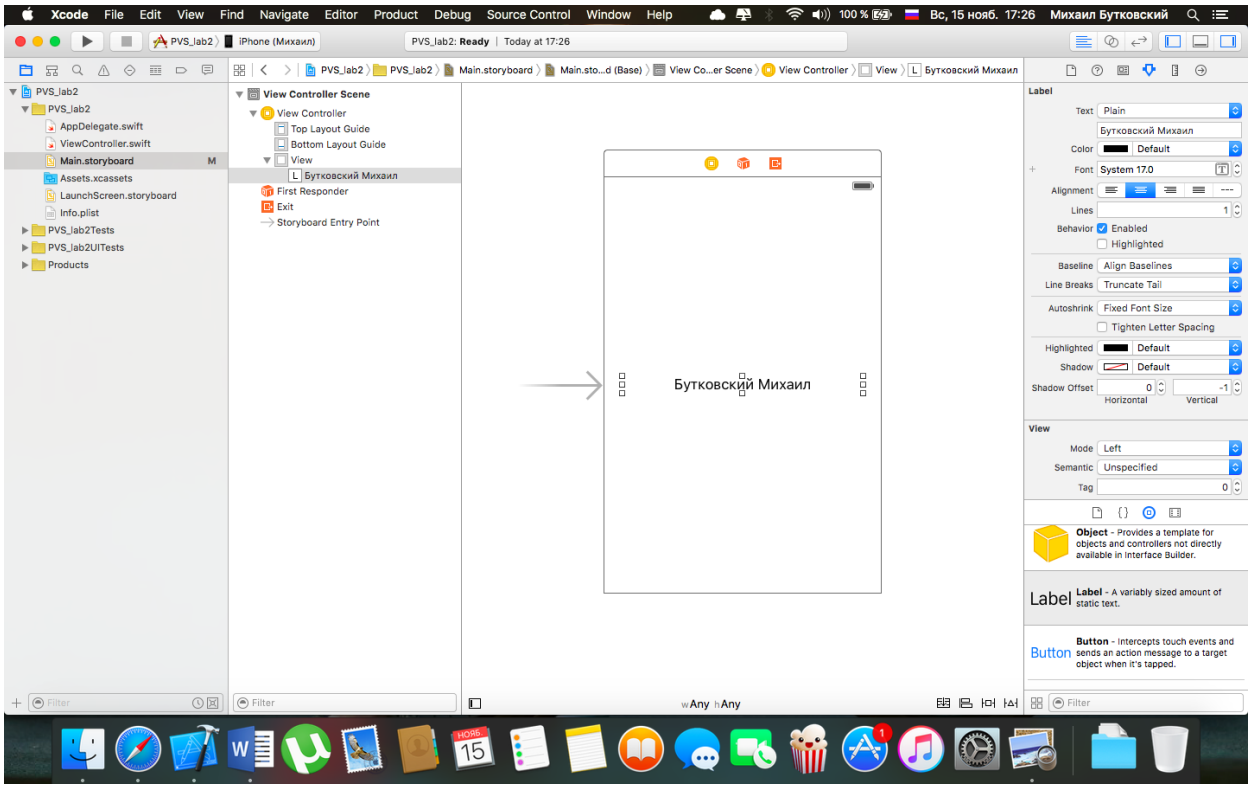
```
@IBAction func pinch(sender: AnyObject)
    {
        gestureIndicator.text = "Жест: масштабирование\n Цвет фона: крас-
```

```
ный"
        gestureIndicator.backgroundColor = UIColor.redColor()
    }
        Для жеста вращения:
        @IBAction func rotation(sender: AnyObject)
    {
        gestureIndicator.text = "Жест: вращение\n Цвет фона: синий"
        gestureIndicator.backgroundColor = UIColor.blueColor()
    }
        Для жеста смахивания:
        @IBAction func swipe(sender: AnyObject)
    {
        gestureIndicator.text = "Жест: смахивание\n Цвет фона: серый"
        gestureIndicator.backgroundColor = UIColor.lightGrayColor()
    }
        Для жеста долгого нажатия:
        @IBAction func longPress(sender: AnyObject)
    {
        gestureIndicator.text = "Жест: долгое нажатие\n Цвет фона: оранже-
вый"
        gestureIndicator.backgroundColor = UIColor.orangeColor()
    }
```

6.5 Сборка проекта и запуск приложения.




Background image: bg2.jpg
Background image: bg1.jpg


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
Вс, 15 нояб. 17:26 Михаил Бутковский
PVS_lab2 iPhone (Михаил)
PVS_lab2: Ready | Today at 17:26
PVS_lab2 PVS_lab2 Main.storyboard Main.sto...d (Base) View Co...er Scene View Controller View Бутковский Михаил
PVS_lab2
PVS_lab2
AppDelegate.swift
ViewController.swift
Main.storyboard M
Assets.xcassets
LaunchScreen.storyboard
Info.plist
PVS_lab2Tests
PVS_lab2UITests
Products
View Controller Scene
View Controller
Top Layout Guide
Bottom Layout Guide
View
Бутковский Михаил
First Responder
Exit
Storyboard Entry Point
Бутковский Михаил
wAny hAny
Label
Text Plain
Бутковский Михаил
Color Default
Font System 17.0
Alignment
Lines 1
Behavior Enabled
Highlighted
Baseline Align Baselines
Line Breaks Truncate Tail
Autoshrink Fixed Font Size
Tighten Letter Spacing
Highlighted Default
Shadow Default
Shadow Offset 0 -1
Horizontal Vertical
View
Mode Left
Semantic Unspecified
Tag 0
Object - Provides a template for objects and controllers not directly available in Interface Builder.
Label Label - A variably sized amount of static text.
Button Button - Intercepts touch events and sends an action message to a target object when it's tapped.

6.6 Подключение устройства iOS к компьютеру, его настройка и запуск приложения.

Однако, на этом настройка не заканчивается. Для того, чтобы запускать приложения, нам необходимо разрешить нашему iOS устройству запуск приложения от разработчика с нашим Apple ID (то есть, от нас). Для этого подключаем iOS устройство к компьютеру посредством USB-кабеля. В схеме компилятора выбираем наше устройство (в моем случае – “iPhone (Михаил)”) и нажимаем “Build and then run the current scheme”, либо сочетание клавиш “cmd+R”.

Компилятор соберет наш проект и скопирует все необходимые данные на наше устройство. Но, при попытке запуска приложения, устройство выдаст ошибку, о которой говорилось ранее – необходимо заставить телефон доверять этому Apple ID. Для этого, на нашем iOS устройстве переходим в раздел “Настройки” – “Основные” – “Профиль” – “ ’Ваш Apple ID’ ” – “Доверять этому разработчику”. Теперь наше устройство считает программы, созданные с помощью этого Apple ID, надежными.

Далее выходим из настроек, находим наше приложение на экране Home (его название будет соответствовать имени проекта, который вы создавали (в данном случае – “PVS_LP_lab2”) и запускаем его (рис.28).

Рис.28. Запуск приложения

7. ЛАБОРАТОРНАЯ РАБОТА №7 «СОЗДАНИЕ IOS ПРИЛОЖЕНИЯ С ИСПОЛЬЗОВАНИЕМ НЕСКОЛЬКИХ КОНТРОЛЛЕРОВ»

Цель работы:

Научиться добавлять контроллеры и файлы с описанием класса этих контроллеров. Изучить реализацию передачи данных между двумя контроллерами, а, также, научиться создавать переходы с одного контроллера на другой.
Создать анкету в области первого контроллера View со следующими полями:

- Фамилия
- Имя
- Отчество
- Возраст
- Место работы

На втором контроллере View добавить возможность ее редактирования.

Ход работы:

7.1 Создание проекта Xcode.

Создаем проект Xcode, аналогично предыдущим лабораторным работам. Убираем галочку Use Core Data и выбираем iOS – Application – Single View Application.

7.2 Добавление контроллера View и создание файла с описанием его класса.

Находим в библиотеке объектов View Controller и располагаем его рядом с уже созданным контроллером (рис.29).

1.Переходим в File – New – File…

2.Выбираем iOS – Source – Cocoa Touch Class. Жмем Next.

3.Задаем имя класса (Class): “EditViewController”

4.Выбираем подкласс (Subclass of): “UIViewController”

5.Выбираем язык программирования (Language): “Swift”

6.Жмем Next.

Теперь, в папке проекта появился только что созданный нами файл с описанием класса UIViewController (рис.30).

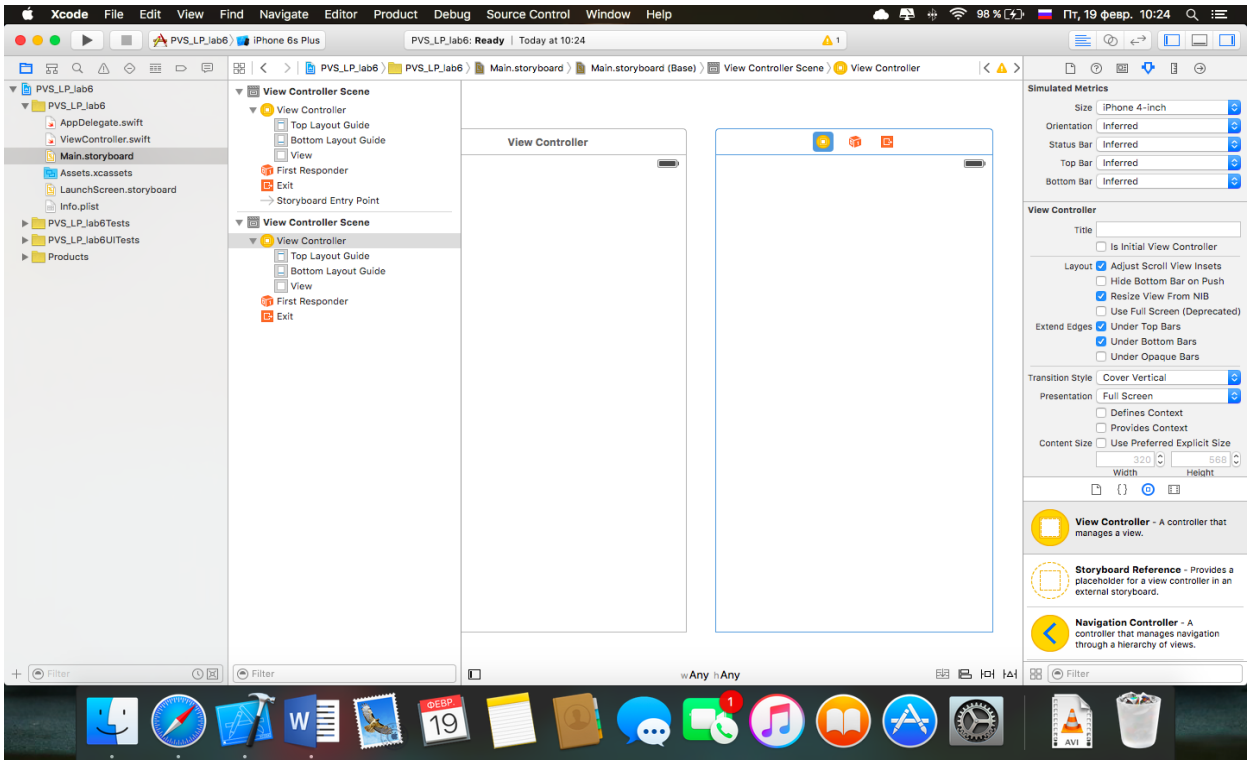


Рис.29. Библиотека объектов View Controller

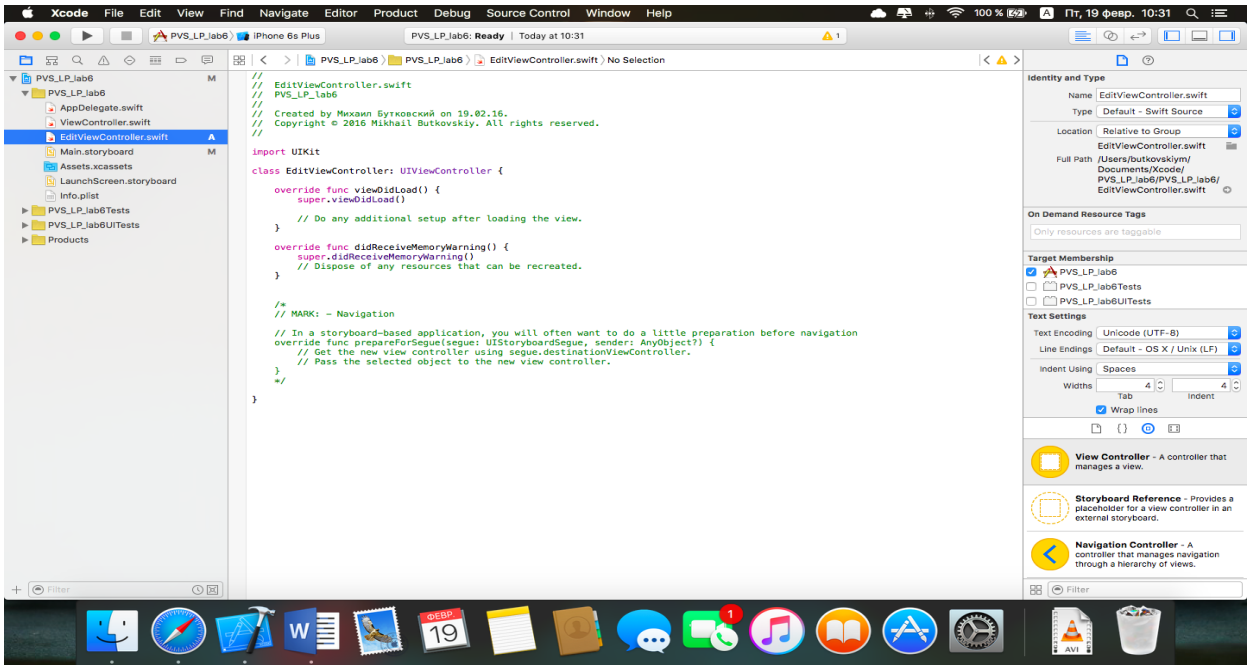


Рис.30. Файл с описанием класса UIViewController

Осталось указать, что данный файл содержит описание класса добавленного нами View контроллера. Переходим в редактор интерфейса. Выбираем добавленный нами контроллер и в его свойствах, во вкладке Show the Identity inspector указываем в поле Class только что созданный нами файл EditViewController (рис.31).

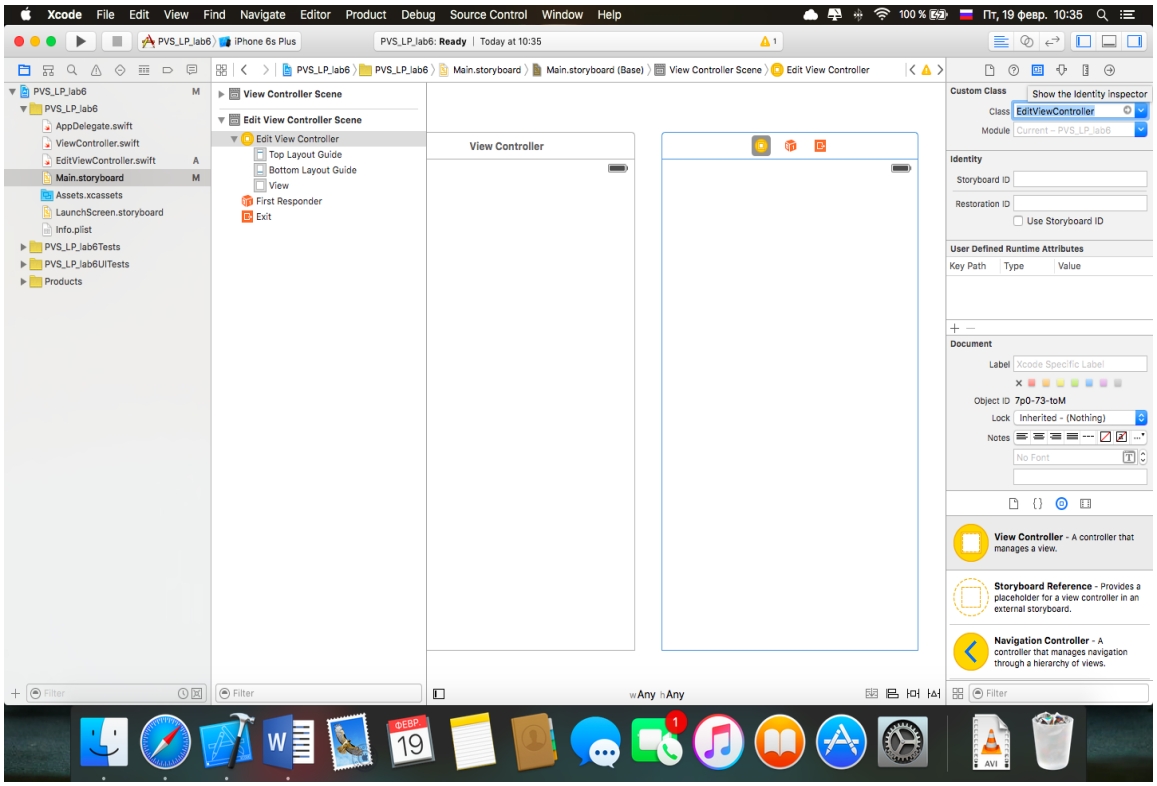


Рис.31. Указание в поле Class в файле.

7.3 Добавление элементов интерфейса и создание связей этих элементов с кодом приложения.

На экран первого контроллера добавляем пять объектов Label с названиями полей анкеты, один с заголовком окна и еще пять пустых (в них будут отображаться данные этих полей). Также, добавляем кнопку редактирования (рис.32).

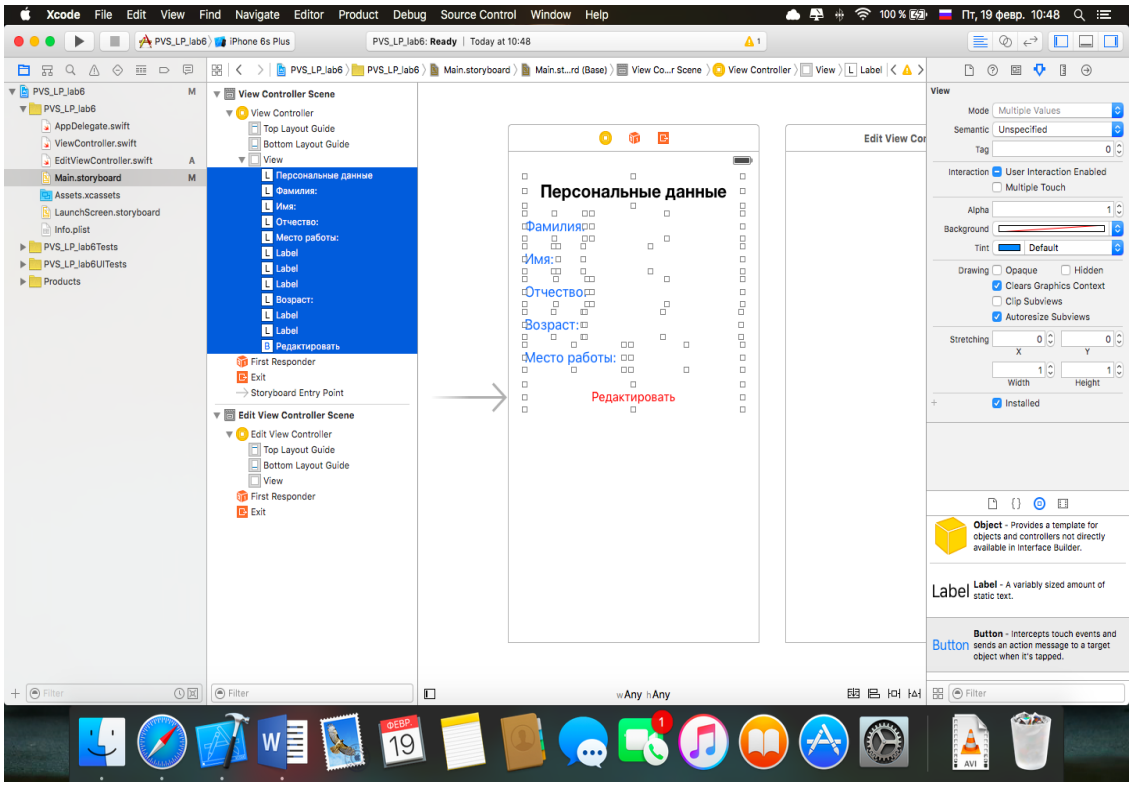


Рис.32. Добавление элементов интерфейса

На экран контроллера редактирования добавляем один объект Label, пять объектов Text Field и один объект Button. В поле Placeholder свойства объектов Text Field указываем названия полей (рис.33).

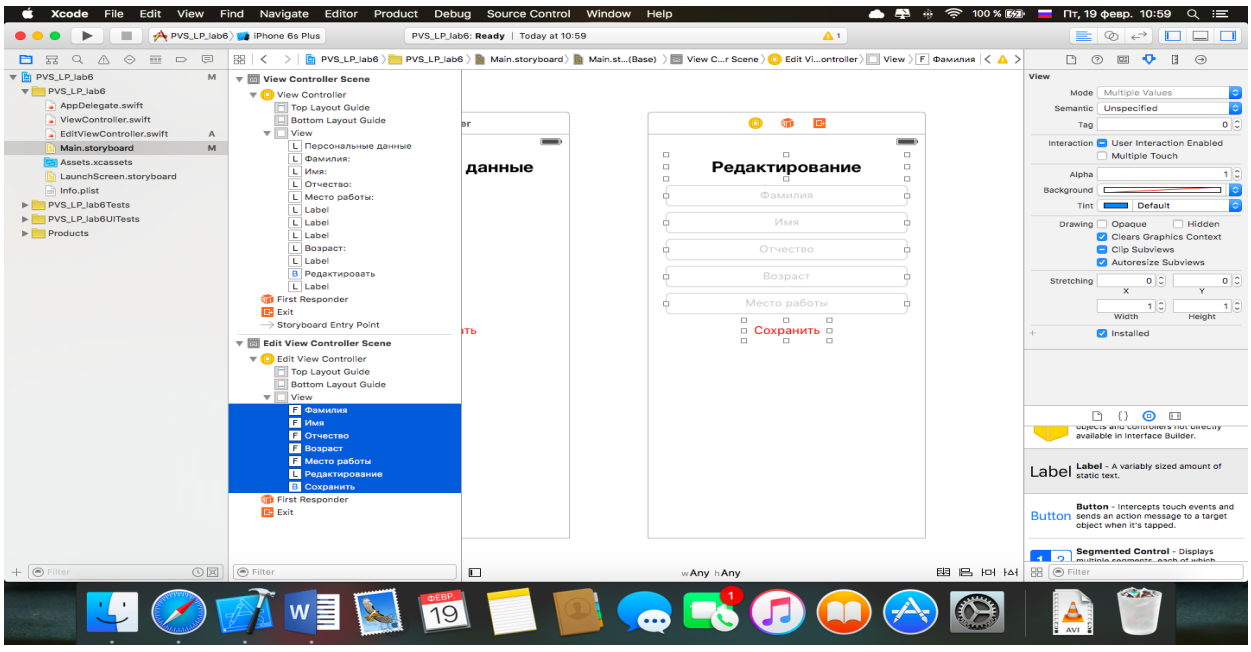


Рис.33. Экран контроллера редактирования

С зажатой клавишей ctrl перетаскиваем объект Button с экрана первого контроллера на экран второго и выбираем Show. Тоже самое делаем с кнопкой на втором контроллере. Тем самым, мы создали две связи. Они означают, что при нажатии на кнопки, приложение будет переходить на другой контроллер (рис.34).

Для пустых объектов Label первого контроллера добавляем связи типа Outlet с файлом ViewController.swift (рис.35).

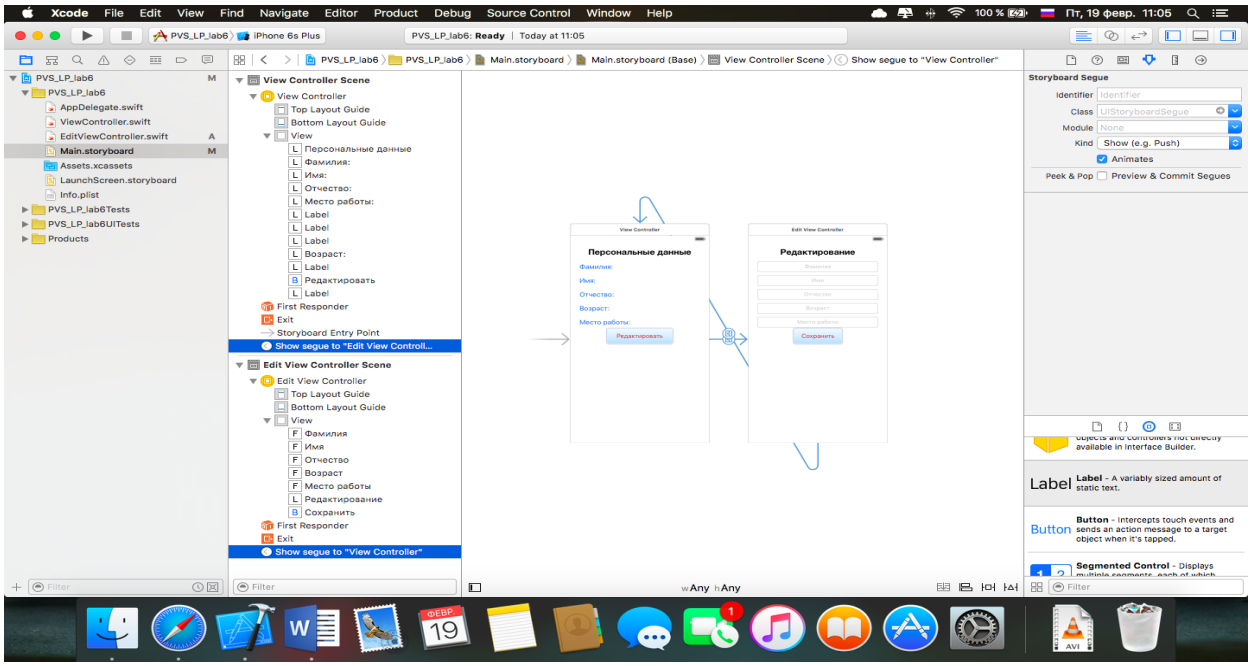


Рис.34. Создание двух связей

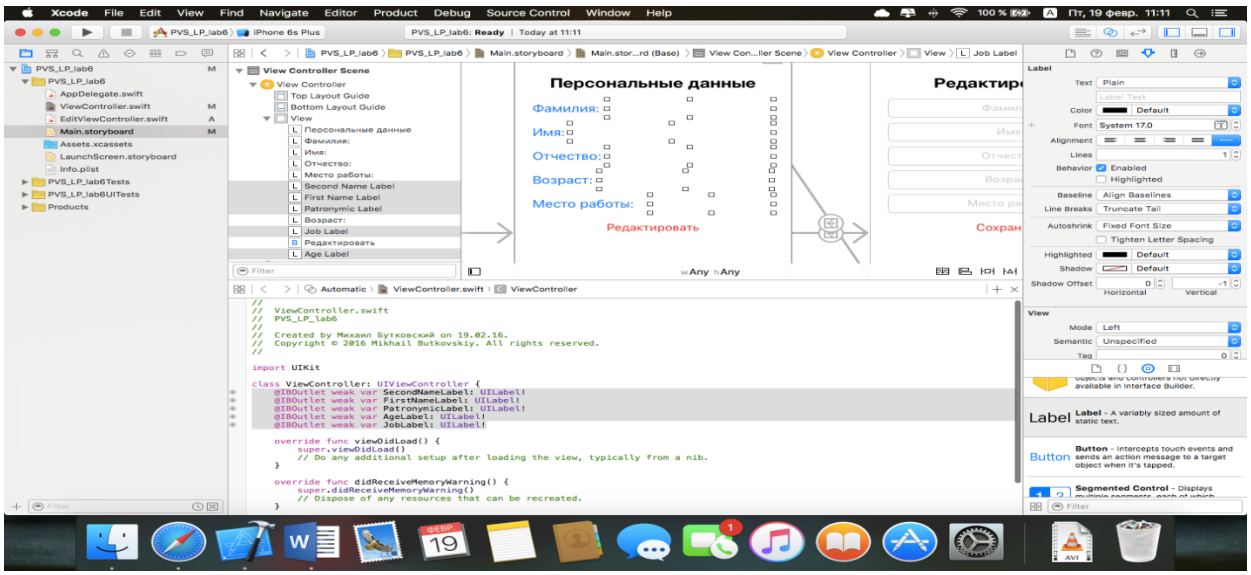


Рис.35. Связи типа Outlet

7.4 Описание событий в коде приложения.

В файле ViewController.swift добавляем пять пустых переменных типа String в описание класса. Они будут использоваться для приема данных с контроллера редактирования.

```
var SecondName = String()
var FirstName = String()
var Patronymic = String()
var Age = String()
var Job = String()
```

В файле EditViewController.swift добавляем функцию перехода на другой контроллер. В ней указываем контроллер, на который будет происходить переход, и данные, которые необходимо передать другому контроллеру (в файл ViewController.swift).

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?)
    {
    let DestinationViewController : ViewController = se-
gue.destinationViewController as! ViewController
    DestinationViewController.FirstName = FirstNameEdit.text!
    DestinationViewController.SecondName = SecondNameEdit.text!
    DestinationViewController.Patronymic = PatronymicEdit.text!
    DestinationViewController.Age = AgeEdit.text!
    DestinationViewController.Job = JobEdit.text!
     }
```

Возвращаемся в файл ViewController.swift и, в функции ViewDidLoad(), описываем прием данных с контроллера редактирования и их присваивание объектам Label (полям анкеты).

```
  override func viewDidLoad()
  {
    super.viewDidLoad()
    FirstNameLabel.text = FirstName
    SecondNameLabel.text = SecondName
    PatronymicLabel.text = Patronymic
```

```
    AgeLabel.text = Age
    JobLabel.text = Job
}
```

7.5 Сборка проекта и запуск приложения.

Выполняем команду cmd+R и проверяем работу приложения (рис.36).

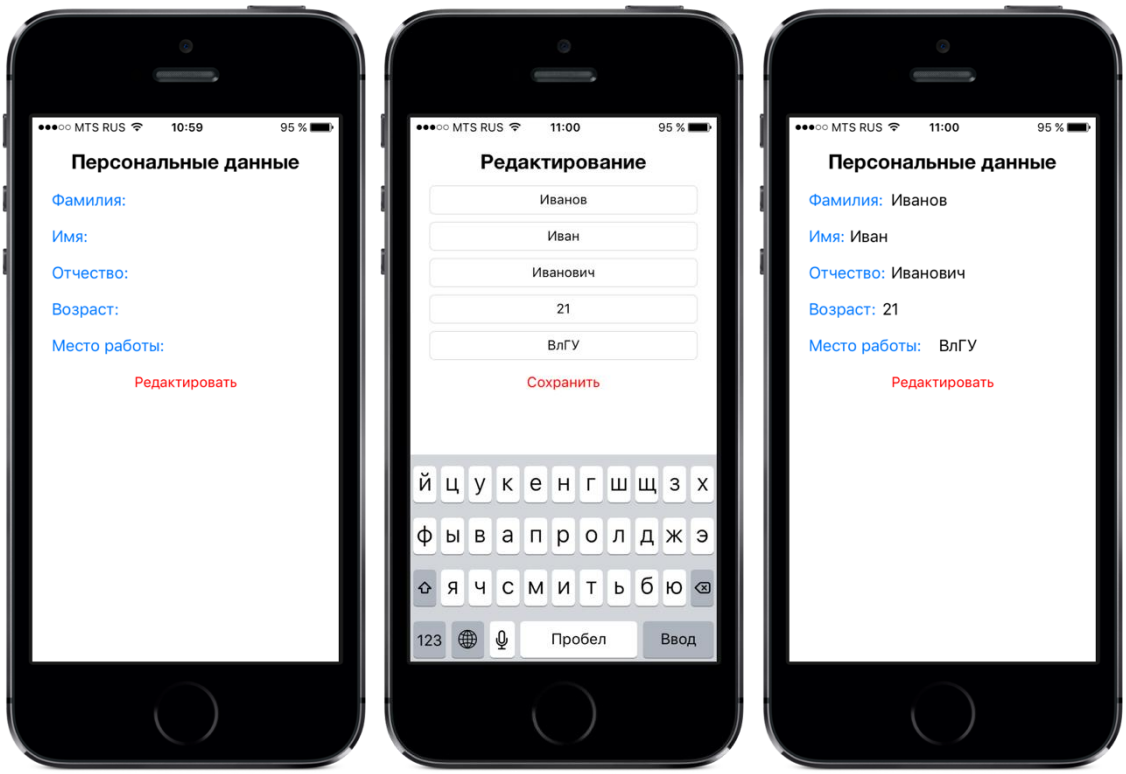


Рис.36. Проверка работы приложения

8. ЛАБОРАТОРНАЯ РАБОТА №8 «СОЗДАНИЕ IOS ПРИЛОЖЕНИЯ С ИСПОЛЬЗОВАНИЕМ КОНТРОЛЛЕРА ТАБЛИЧНОГО ВИДА»

Цель работы:

Создать приложение для работы со списком студентов. Добавить возможность добавления и удаления студента из списка.

Ход работы:

8.1 Создание проекта Xcode.

Создаем проект Xcode, аналогично предыдущим лабораторным работам. Убираем галочку Use Core Data и выбираем iOS – Application – Single View Application.

8.2 Разработка интерфейса приложения.

Находим в библиотеке объектов контроллер Table View Controller и располагаем его рядом с уже созданным контроллером View (рис.37).

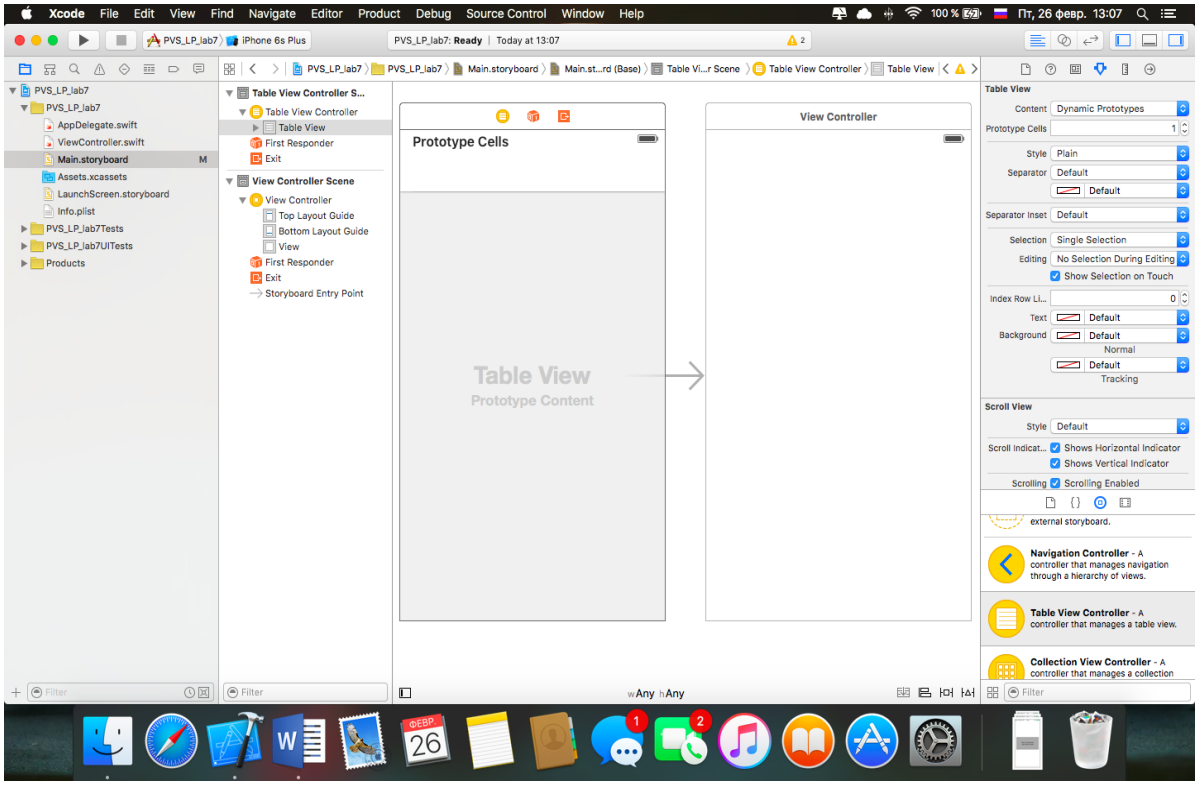


Рис.37. Библиотека объектов

Теперь подключим к нему Navigation Controller, который обеспечит работу переходов и создаст контроллеру Table View навигационную строку (Navigation Bar). Выбираем его и переходим в Editor – Embed In – Navigation Controller. Установим контроллер Navigation как входной контроллер приложения (галочка Is Initial View Controller в его свойствах) (рис.38).

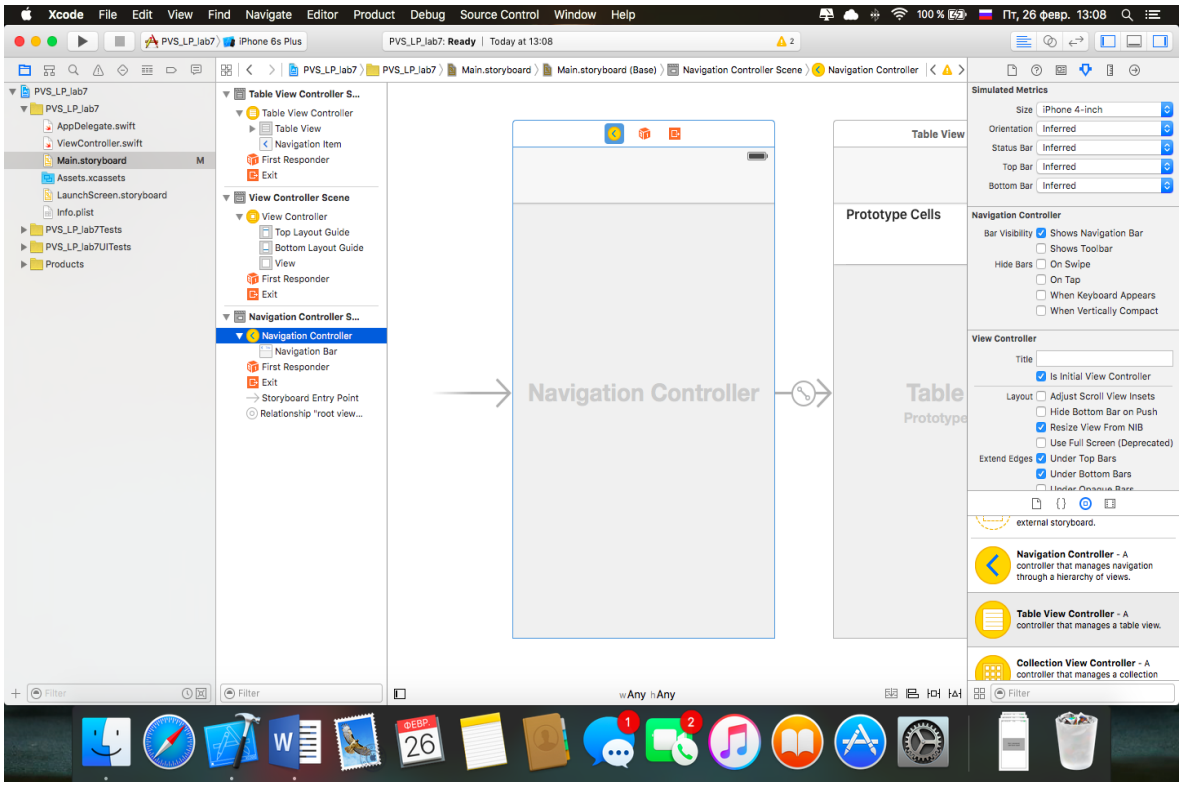


Рис.38. Установка контроллера Navigation

В контроллере Table View установим заголовок для навигационной строки: “Список студентов” и добавим кнопку добавления нового студента Bar Button Item, поменяв ее название на: “Добавить”. Также, создадим переход на контроллер View от кнопки “Добавить”, перетащив ее с зажатой клавишей ctrl и выбрав Present Modaly. Такой способ перехода означает, что контроллер View откроется поверх контроллера Table View (рис.39).

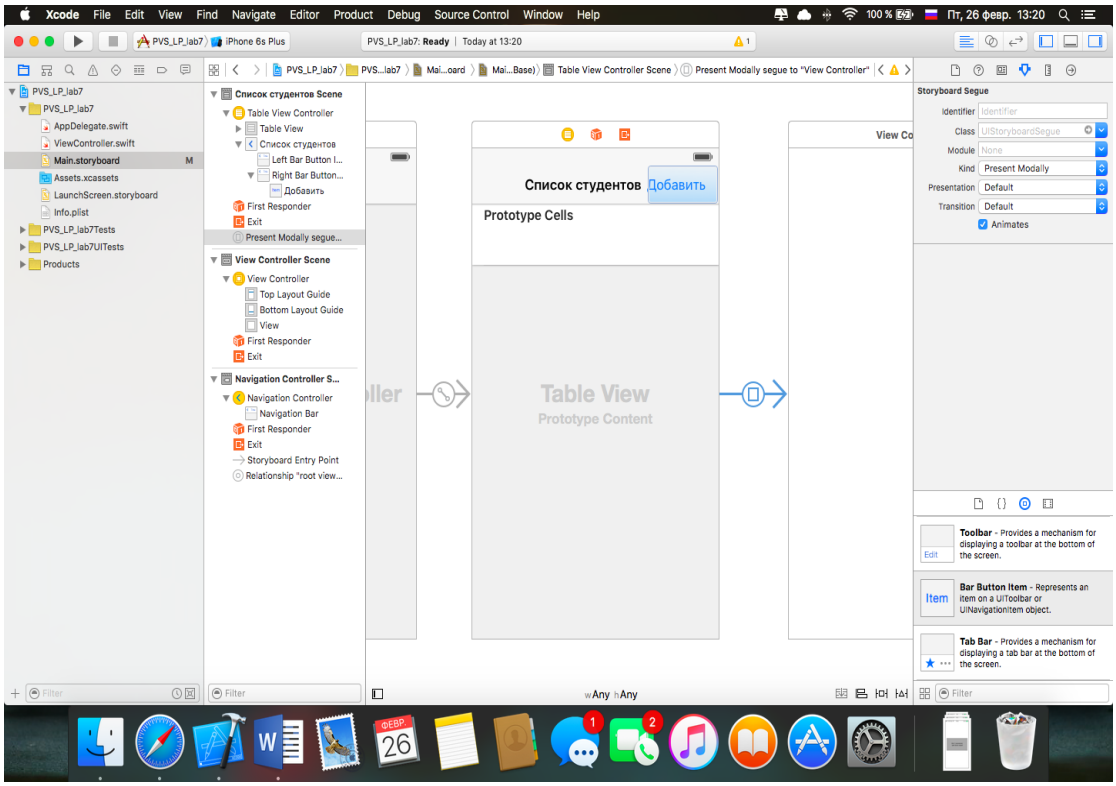


Рис.39. Создание перехода на контроллер View от кнопки “Добавить”

В свойствах ячейки Cell таблицы Table View контроллера Список студентов укажем идентификатор (поле Identifier): studentCell (рис.40).

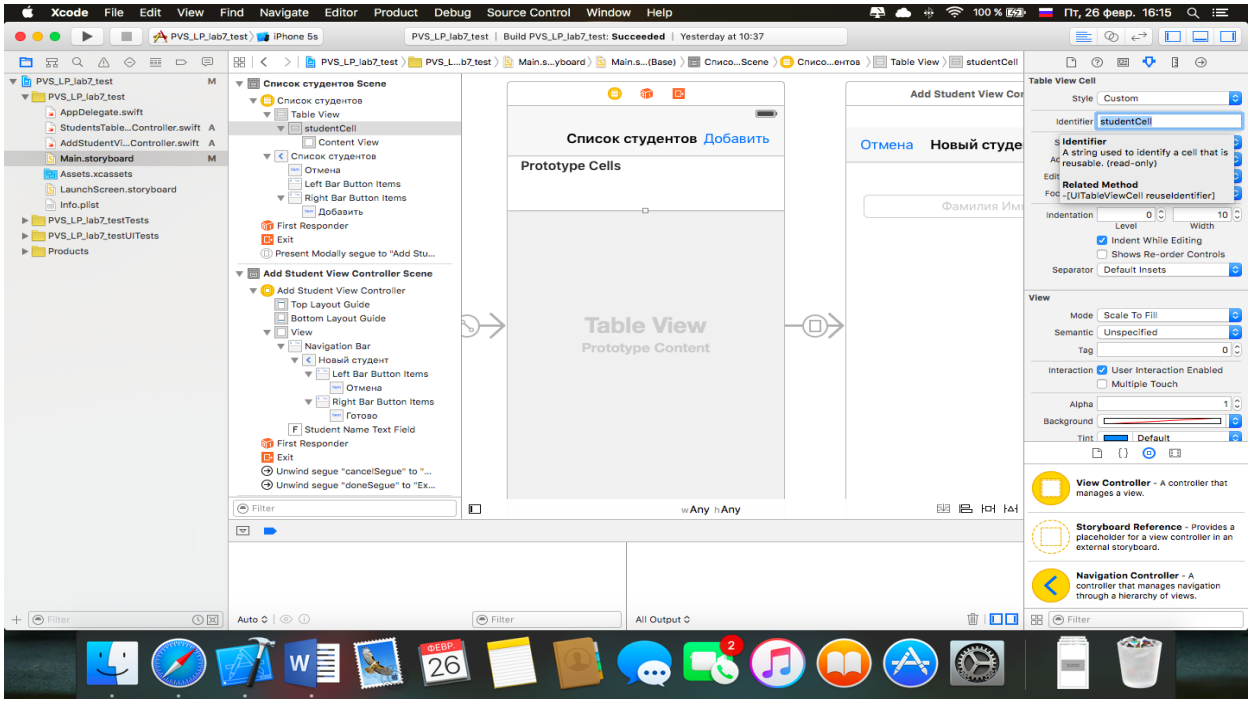


Рис.40. Свойства ячейки Cell таблицы Table View

В контроллере View добавим Navigation Bar, две кнопки Bar Button Item и текстовое поле Text Field, у которого зададим Placeholder: “Фамилия Имя” в свойствах (рис.41).

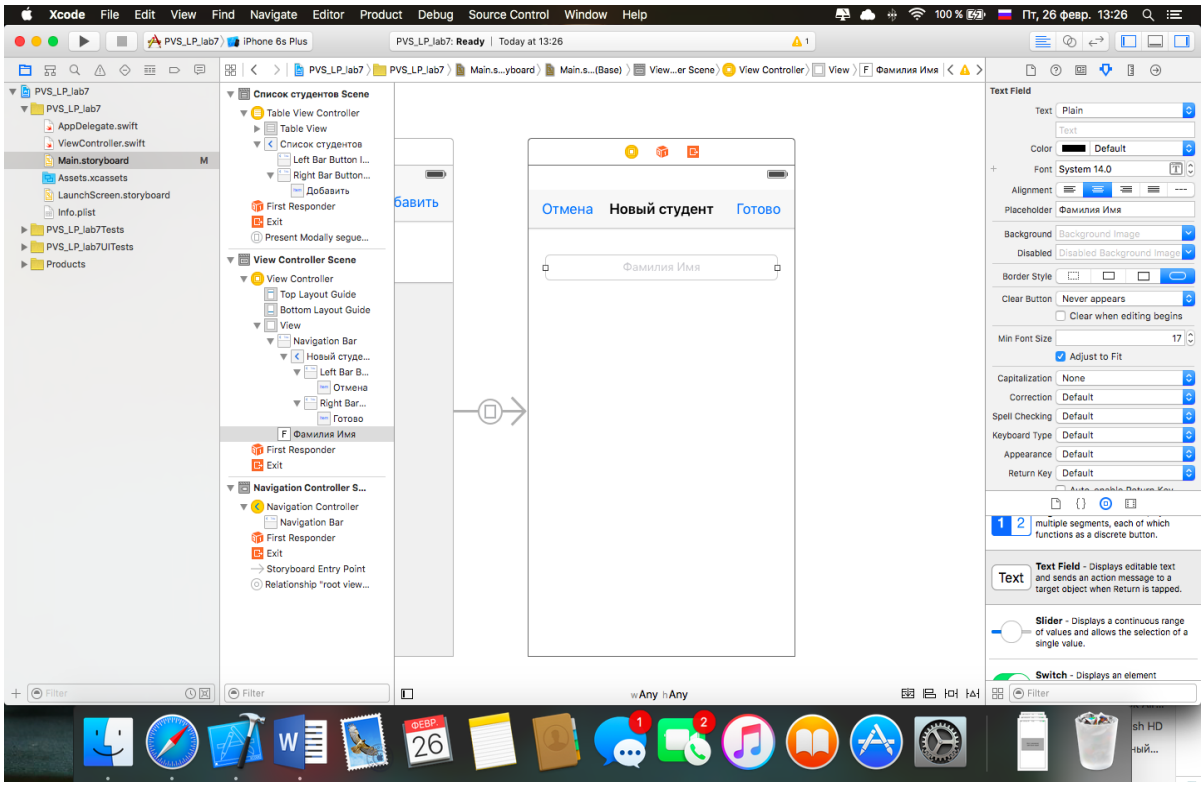


Рис.41.Добавление компонентов в контроллер View

8.3 Создание связей объектов интерфейса с кодом приложения и описание событий в коде приложения.

Создаем файл с описанием класса контроллера Table View, аналогично предыдущей лабораторной работе. В описании класса добавляем массив, который будет хранить список студентов и переменную, которая будет принимать значение с контроллера View, при добавлении нового студента:

```
var students = [String]()
var newStudent:String = ""
```

Заполняем метод, возвращающий количество секций в таблице:

```
override func numberOfSectionsInTableView(tableView: UITableView) -> Int
    {
        return 1
    }
```

Заполняем метод, возвращающий количество строк в секции таблицы, равное количеству элементов массива students:

```
override func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int
  {
    return students.count
  }
```

Заполняем метод, возвращающий значение ячейке, используя идентификатор, который необходимо указать в свойствах таблицы Table View. Значение ячейки должно быть равным значению элемента массива students:

```
override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
  {
    let cell = tableView.dequeueReusableCellWithIdentifier("studentCell", forIndexPath: indexPath)
    cell.textLabel!.text = students[indexPath.row]
    return cell
  }
```

Создаем метод редактирования ячеек, в котором будет происходить удаление студента из списка. Пишем условие, которое означает, что при выборе удаления, как стиля редактирования ячейки, будет происходить удаление выбранного элемента из массива и ячейки из таблицы.

```
override func tableView(tableView: UITableView, commitEditingStyle editingStyle: UITableViewCellEditingStyle, forRowAtIndexPath indexPath: NSIndexPath)
  {
    if editingStyle == UITableViewCellEditingStyle.Delete
    {
      students.removeAtIndex(indexPath.row)
      tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation: UITableViewRowAnimation.Left)
    }
  }
```

Также, добавляем два пустых события @IBAction, которые в дальнейшем мы будем использовать с кнопками “Готово” и “Отмена” на контроллере “Добавление студента”.

```
@IBAction func done(segue: UIStoryboardSegue)
    {

    }

    @IBAction func cancel(segue: UIStoryboardSegue)
    {

    }
```

Открываем редактор интерфейса и, с зажатой клавишей ctrl, перетаскиваем кнопки “Готово” и “Отмена” на значок Exit контроллера AddStudentViewController, чтобы, при нажатии на них, контроллер закрывался. Для кнопки “Готово” выбираем созданный ранее метод done, а для кнопки “Отмена” – метод cancel (рис.42).

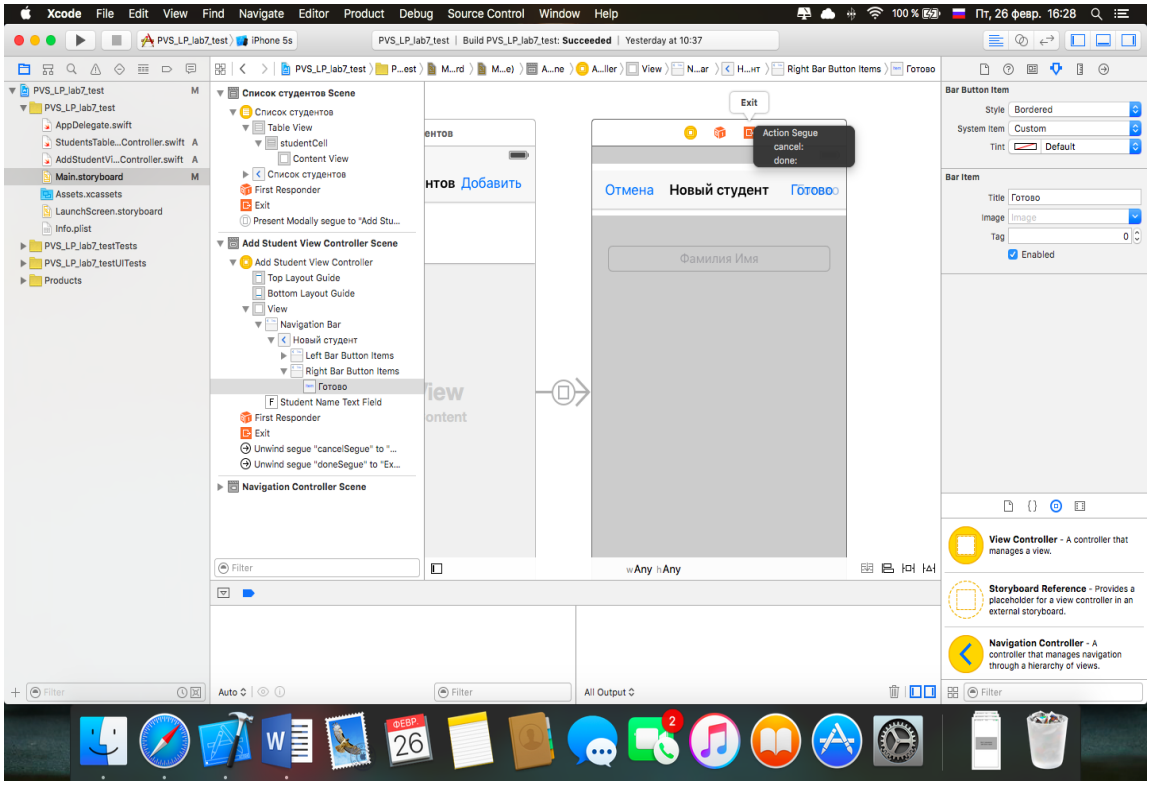


Рис.42. Редактор интерфейса

Переходим в файл AddStudentViewContoller.swift.
Создаем связь типа Outlet с объектом Text Field:

```
@IBOutlet weak var studentNameTextField: UITextField!
```

Создаем переменную, в которую будем заносить Фамилию и Имя нового студента для передачи переменной newStudent (файл StudentsTableViewController.swift):

```
var studentNameToAdd:String = ""
```

В дереве объектов интерфейса находим переход, подключенный к методу done и задаем ему идентификатор doneSegue (рис.43).

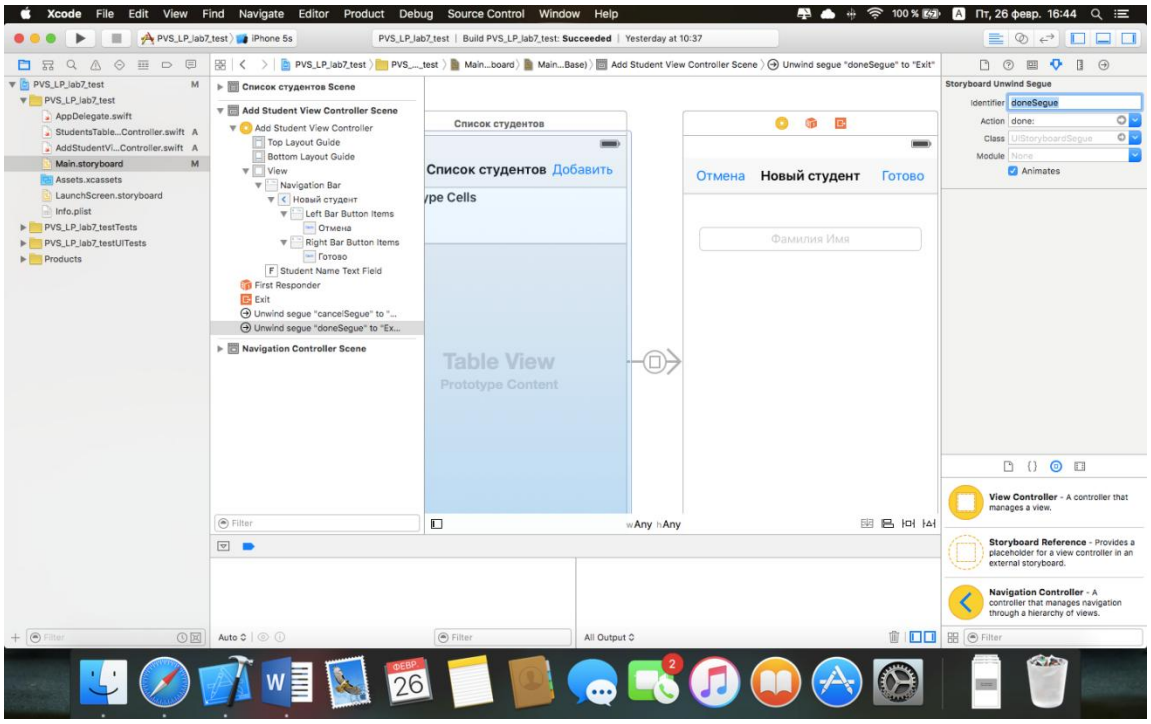


Рис.43. Дерево объектов интерфейса

Создаем функцию для подготовки передачи данных. В ней присваиваем переменной studentNameToAdd текст из поля Фамилия Имя, если использован переход с идентификатором doneSegue (нажата кнопка Готово):

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?)
    {
        if segue.identifier == "doneSegue"
        {
            studentNameToAdd = studentNameTextField.text!
        }
    }
```

Возвращаемся в файл StudentsTableViewController.swift и заполняем метод done, в котором указываем AddStudentViewController, как контроллер, с которого необходимо принять информацию (из метода prepareForSegue), присваиваем значение переменной с Фамилией и Именем студента из текстового поля переменной newStudent, добавляем ее в массив и перезагружаем данные таблицы, чтобы в ней отобразилось текущее содержимое массива с новыми данными:

```
@IBAction func done(segue: UIStoryboardSegue)
    {
        let addStudentNameVC = segue.sourceViewController as! AddStudentView-
Controller
        newStudent = addStudentNameVC.studentNameToAdd
        students.append(newStudent)
        self.tableView.reloadData()
        print(students)
    }
```

Функцию cancel оставляем пустой, т.к., при нажатии на кнопку Отмена не должно происходить никаких действий, кроме закрытия контроллера:

```
    @IBAction func cancel(segue: UIStoryboardSegue)
    {

    }
```

8.4 Сборка проекта и запуск приложения.

Выполняем команду cmd+R и проверяем работу приложения (рис.44).

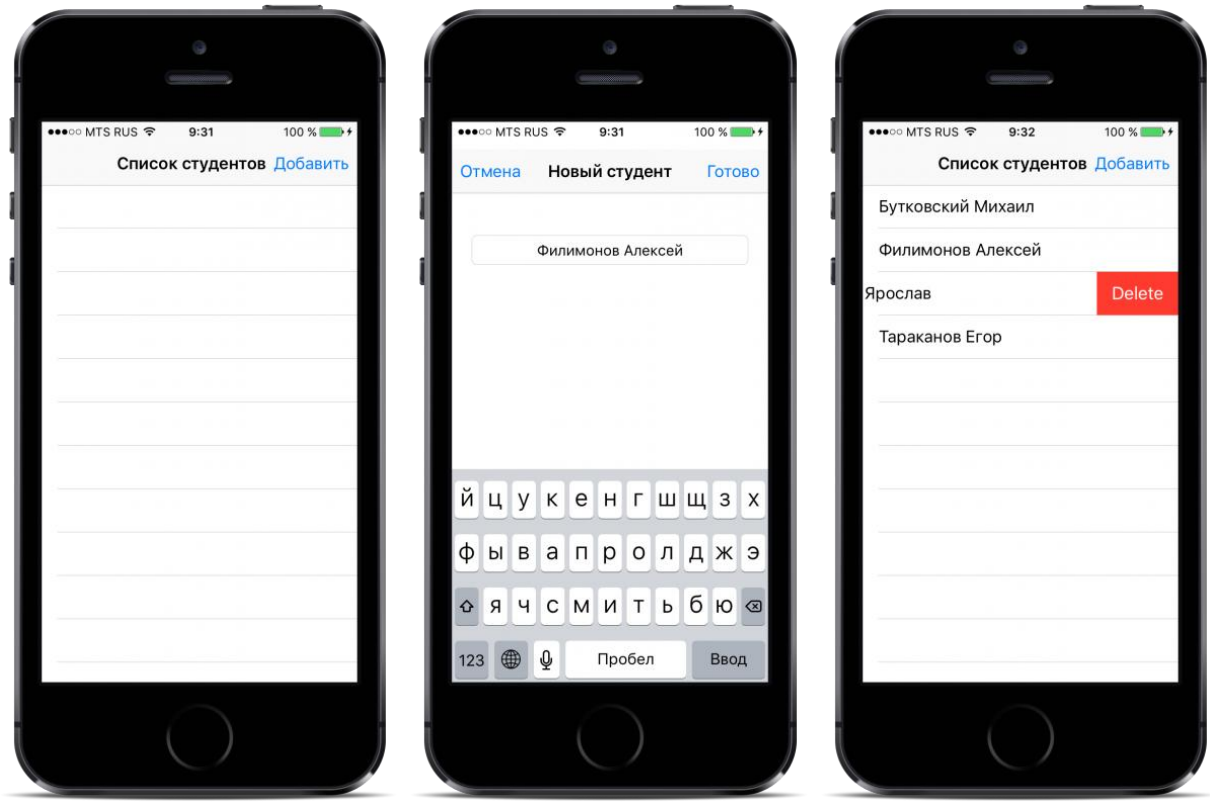


Рис.44. Проверка работы приложения

9. ЛАБОРАТОРНАЯ РАБОТА №9 «СОЗДАНИЕ IOS ПРИЛОЖЕНИЯ С ИСПОЛЬЗОВАНИЕМ БАЗЫ ДАННЫХ CORE DATA»

Цель работы:

Научиться работать со встроенными в Xcode средствами для работы с базами данных – Core Data. Научиться добавлять, удалять и получать объекты из базы данных.

Задание к работе:

Создать список студентов, который будет храниться в базе данных. При закрытии приложения, данные должны сохраняться. Реализовать возможность добавления и удаления студентов из списка. Список должен быть отображен в объекте Table View. Добавление должно происходить с помощью объекта Text Field по кнопке Button.

Ход работы:

9.1 Создание проекта Xcode.

Создаем проект iOS – Application – Single View Application, но, на этот раз, ставим галочку напротив Use Core Data (рис.45).

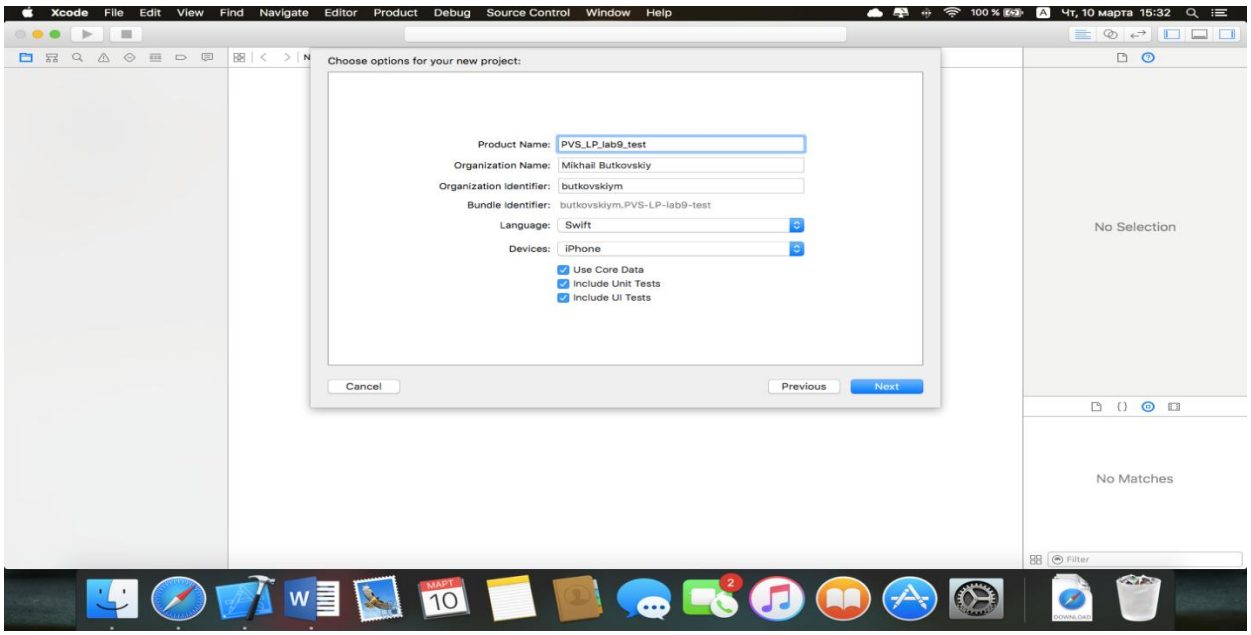


Рис.45. Создание проекта

Тем самым, Xcode автоматически добавит в описание класса AppDelegate в файле AppDelegate.swift все необходимые методы для работы с базами данных. На них мы и будем ссылаться (рис.46).

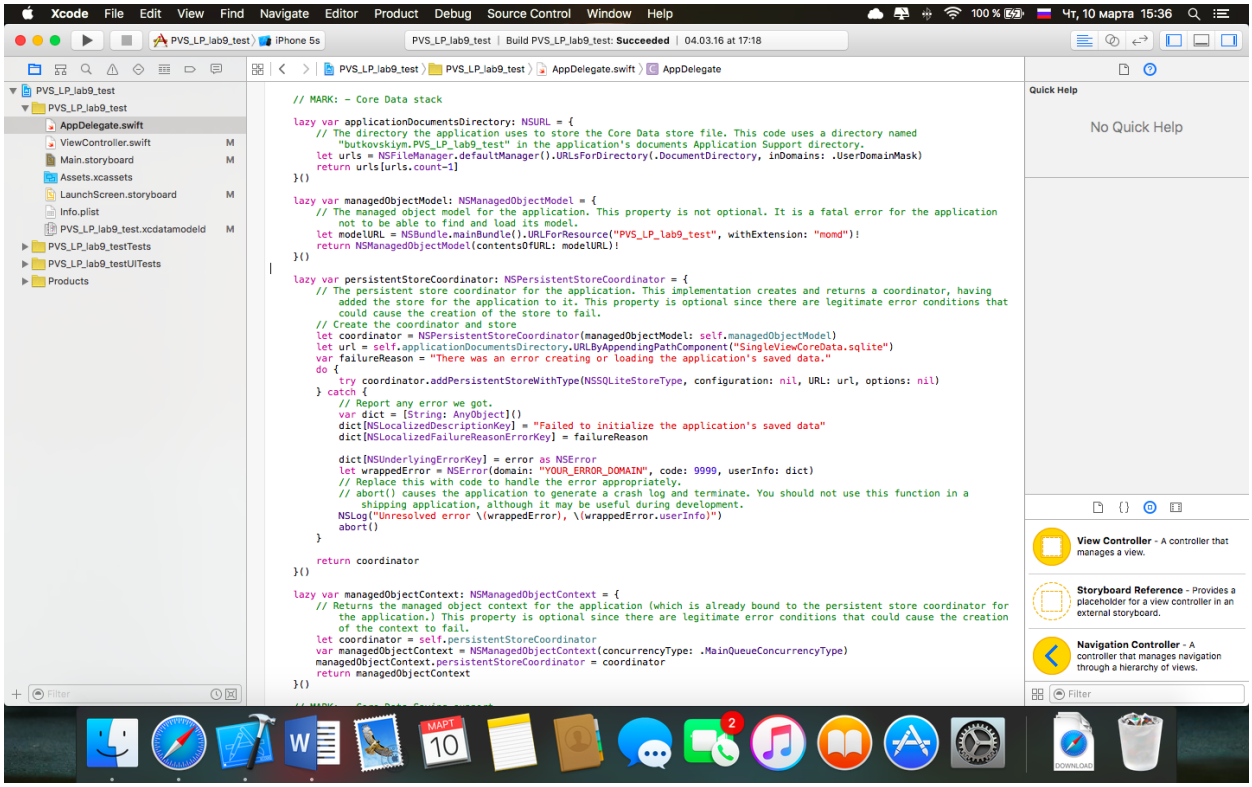


Рис.46. Описание класса AppDelegate

9.2 Добавление элементов интерфейса. Создание связей элементов интерфейса с кодом приложения.

Добавляем объект Label для отображения заголовка и таблицу Table View, внутрь которой добавляем ячейку Table View Cell. Также, добавляем текстовое поле Text Field, куда пользователь будет вводить Фамилию и Имя нового студента и кнопку Button, при нажатии на которую, новый студент будет добавлен в таблицу (рис.47). Ячейке Table View Cell присваиваем идентификатор: “studentCell”. Текстовому полю задаем Placeholder: “Введите Фамилию и Имя студента”.

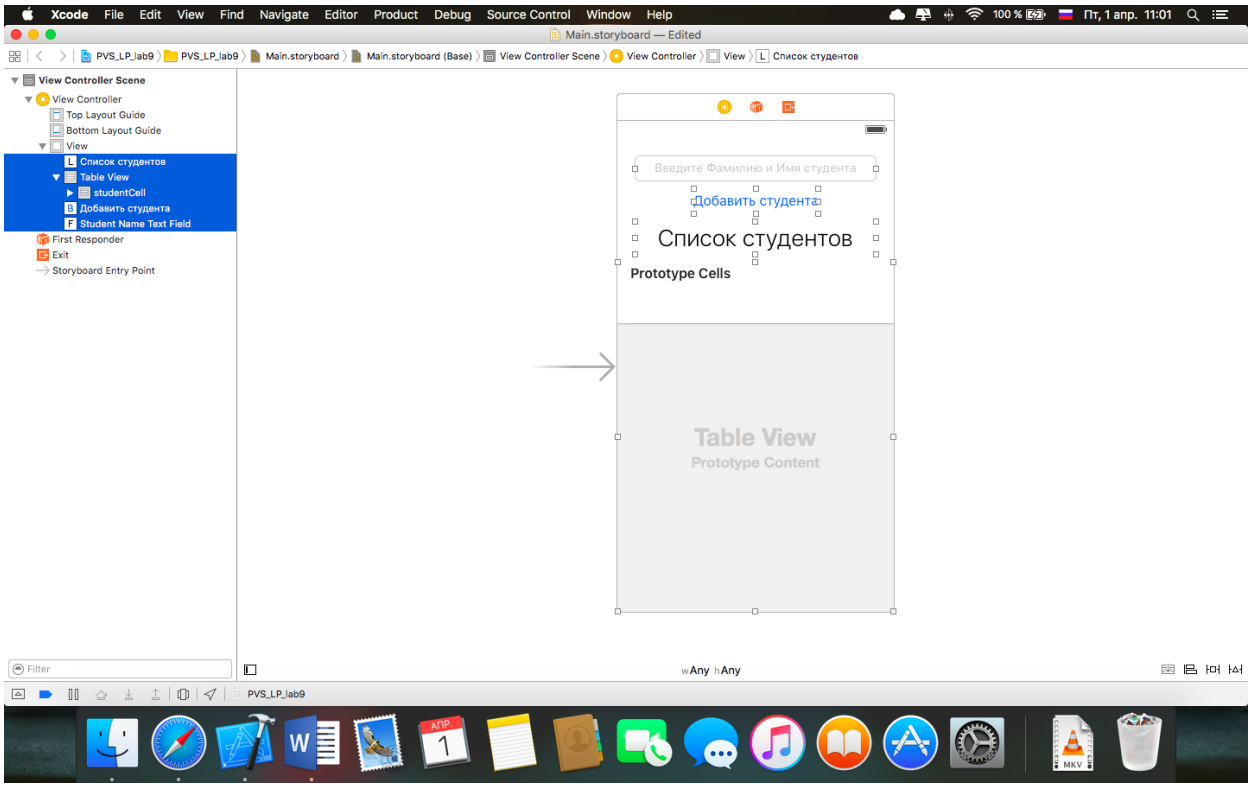


Рис.47. Добавление объекта Label

Перетаскиваем объект Table View на View Controller в дереве объектов интерфейса и выбираем dataSource для того, чтобы работать с данными этой таблицы в описании класса ViewController (рис.48).

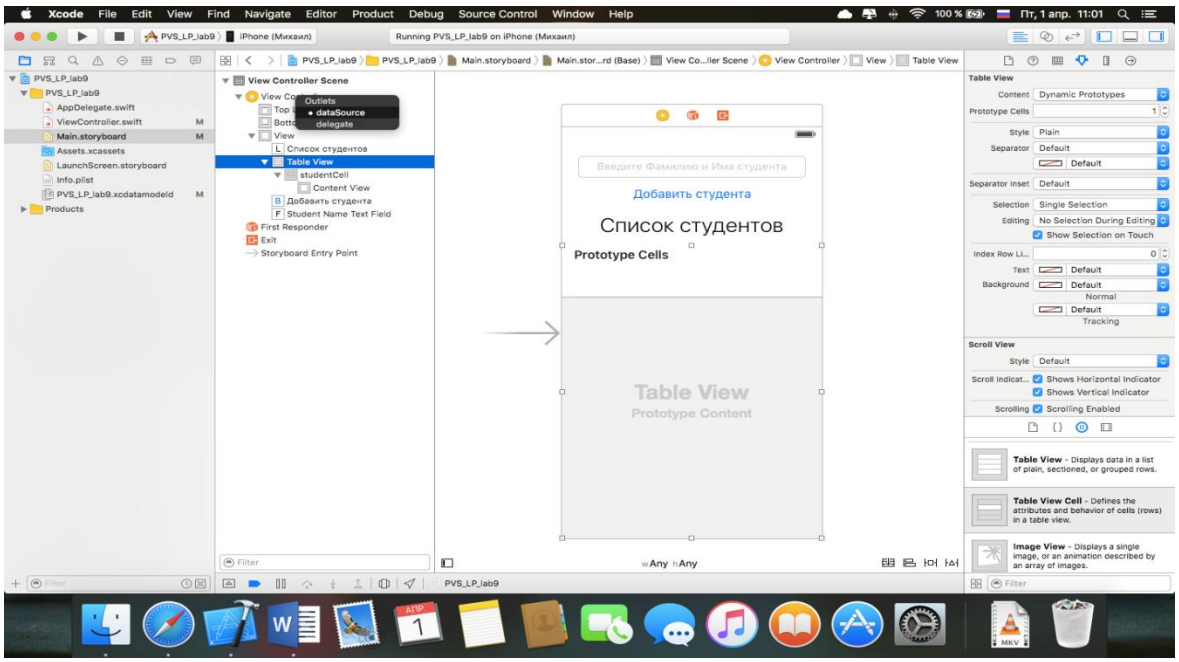


Рис.48. Дерево объектов интерфейса

Для объектов Text Field и Table View создаем Outlet-соединения с описанием класса View Controller:

```
@IBOutlet weak var studentNameTextField: UITextField!
@IBOutlet weak var tableView: UITableView!
```

Для объекта Button создаем Action-соединение (метод):

```
@IBAction func addStudentButton(sender: AnyObject){
}
```

9.3 Создание модели базы данных.

Открываем файл с расширением .xcdatamodeld и добавляем сущность Students, нажав внизу на кнопку Add Entity. В сущность Students добавляем атрибут “name” типа String (рис.49).

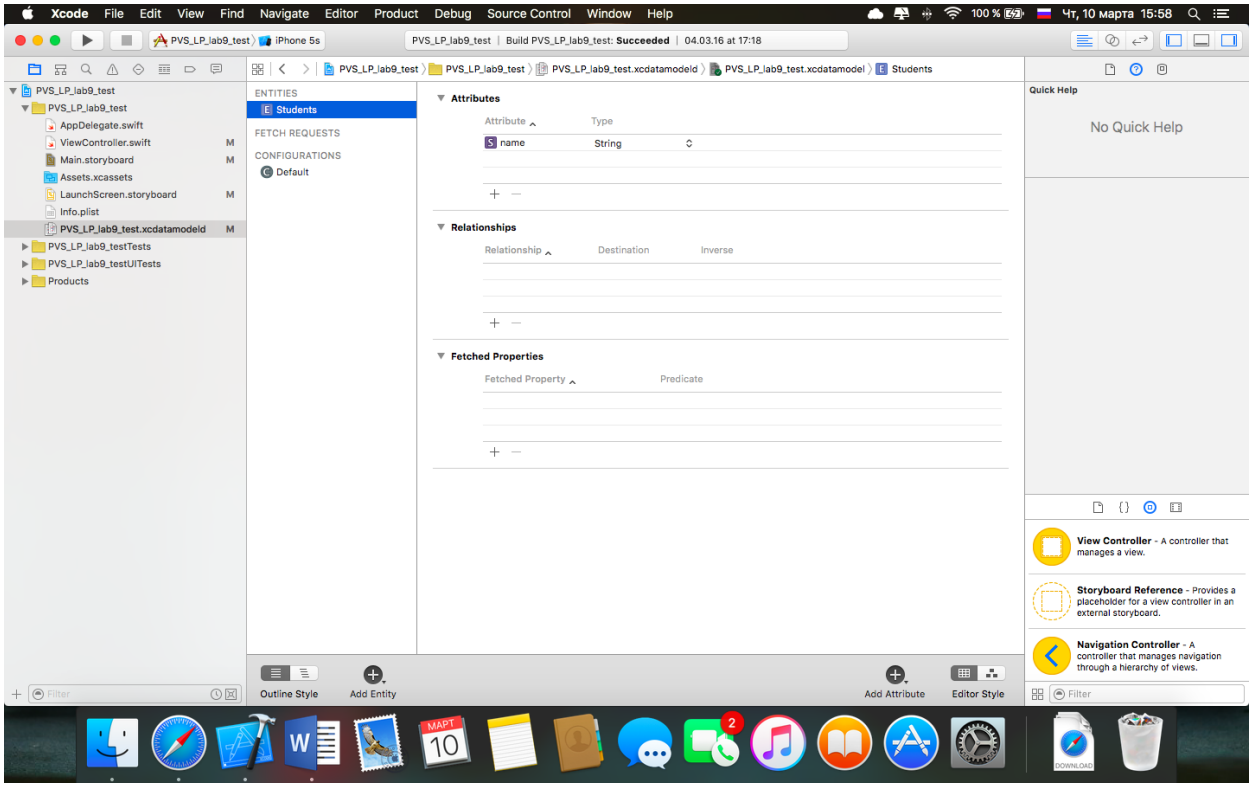


Рис.49. Файл с расширением .xcdatamodeld

9.4 Описание событий в коде приложения.

Подключаем к файлу ViewController.swift библиотеку для работы с базами данных:

```
import CoreData
```

Подписываем класс ViewController под протокол UITableViewDataSource:

```
class ViewController: UIViewController, UITableViewDataSource
```

В описании класса ViewController.swift объявляем пустой массив класса NSManagedObject, который будет являться моделью наших данных:

```
var students = [NSManagedObject]()
```

Задаем количество секций для таблицы:

```
    func numberOfSectionsInTableView(tableView: UITableView) -> Int
    {
  return 1
    }
```

Задаем количество строк в секции таблицы, равное количеству объектов массива students:

```
    func tableView(tableView: UITableView, numberOfRowsInSection section: Int)
-> Int
    {
  return students.count
    }
```

Задаем значение для ячейки таблицы:

```
  func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath:
NSIndexPath) -> UITableViewCell
  {
  let cell = tableView.dequeueReusableCellWithIdentifier("studentCell")! as
UITableViewCell

  let student = students[indexPath.row]
  cell.textLabel?.text = student.valueForKey("name") as? String //Заполняем
текст ячейки таблицы значением ключа "name"

  return cell
  }
```

Создаем метод для удаления элемента из таблицы и из базы данных:

```
  func tableView(tableView: UITableView, commitEditingStyle editingStyle: UITa-
```

```
bleViewCellEditingStyle, forRowAtIndexPath indexPath: NSIndexPath)
{
    if editingStyle == UITableViewCellEditingStyle.Delete
    {
        let appDelegate = UIApplication.sharedApplication().delegate as! AppDelegate //Создаем ссылку на класс AppDelegate из файла AppDelegate.swift
        let managedObjectContext = appDelegate.managedObjectContext //Создаем ссылку на метод managedObjectContext из класса AppDelegate в файле AppDelegate.swift

        managedObjectContext.deleteObject(students[indexPath.row] as NSManagedObject) //Выбираем метод удаления объекта из модели данных students
        do
        {
            try managedObjectContext.save() //Пробуем сохранить изменения в базе данных
            students.removeAtIndex(indexPath.row) //Удаляем объект из модели students
            tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation: .Left) //Удаляем строку из таблицы
        }
        catch let error as NSError
        {
            print("Data removing error: \(error)") //В случае возникновения ошибки, выводим ее в консоль
        }
    }

}
```

Заполняем Action-метод для кнопки “Добавить студента”:

```
@IBAction func addStudentButton(sender: AnyObject)
{
    if studentNameTextField.text == "" || studentNameTextField.text == "Введите данные!"
    {
        studentNameTextField.text = "Введите данные!"
    }
```

```
        else
        {
            let appDelegate = UIApplication.sharedApplication().delegate as! AppDelegate //Создаем ссылку на класс AppDelegate из файла AppDelegate.swift
            let managedObjectContext = appDelegate.managedObjectContext //Создаем ссылку на объект managedObjectContext из класса AppDelegate

            let newObject = NSEntityDescription.insertNewObjectForEntityForName("Students", inManagedObjectContext: managedObjectContext) as NSManagedObject //Создаем переменную, которая будет заносить новый объект в сущность "Students"

            newObject.setValue(studentNameTextField.text!, forKey: "name") //Значением нового объекта для ключа "name" будет являться текст из текстового поля studentNameTextField

            do
            {
                try managedObjectContext.save() //Пробуем сохранить изменения в модели
                students.append(newObject) //Добавляем новый объект в модель данных students
                studentNameTextField.text! = "" //Очищаем текстовое поле
                self.tableView.reloadData() //Обновляем содержимое таблицы
self.view.endEditing(true) //Убираем клавиатуру с экрана и выходим из режима редактирования
            }
            catch let error as NSError
            {
                print("Data saving error: \(error)") //В случае возникновения ошибки, выводим ее в консоль
            }
        }
}
```

Создаем метод viewWillAppear, который запускается при загрузке области View в контроллере (в этой области расположен объект Table View):

```
override func viewWillAppear(animated: Bool)
{
    super.viewWillAppear(animated)

    let appDelegate = UIApplication.sharedApplication().delegate as! AppDelegate
//Создаем ссылку на класс AppDelegate из файла AppDelegate.swift
    let managedObjectContext = appDelegate.managedObjectContext //Создаем
ссылку на объект managedObjectContext из класса AppDelegate

    let fetchRequest = NSFetchRequest(entityName: "Students") //Создаем запрос
из сущности "Students"

    do
    {
      students = try managedObjectContext.executeFetchRequest(fetchRequest) as!
[NSManagedObject] //Пробуем загрузить запрошенные данные в модель students
    }
    catch let error as NSError
    {
      print("Data loading error: \(error)")
    }
    self.tableView.reloadData() //Обновляем содержимое таблицы
}
```

9.5 Сборка проекта и запуск приложения.

Выполняем команду cmd+R и проверяем работу приложения (рис.50).

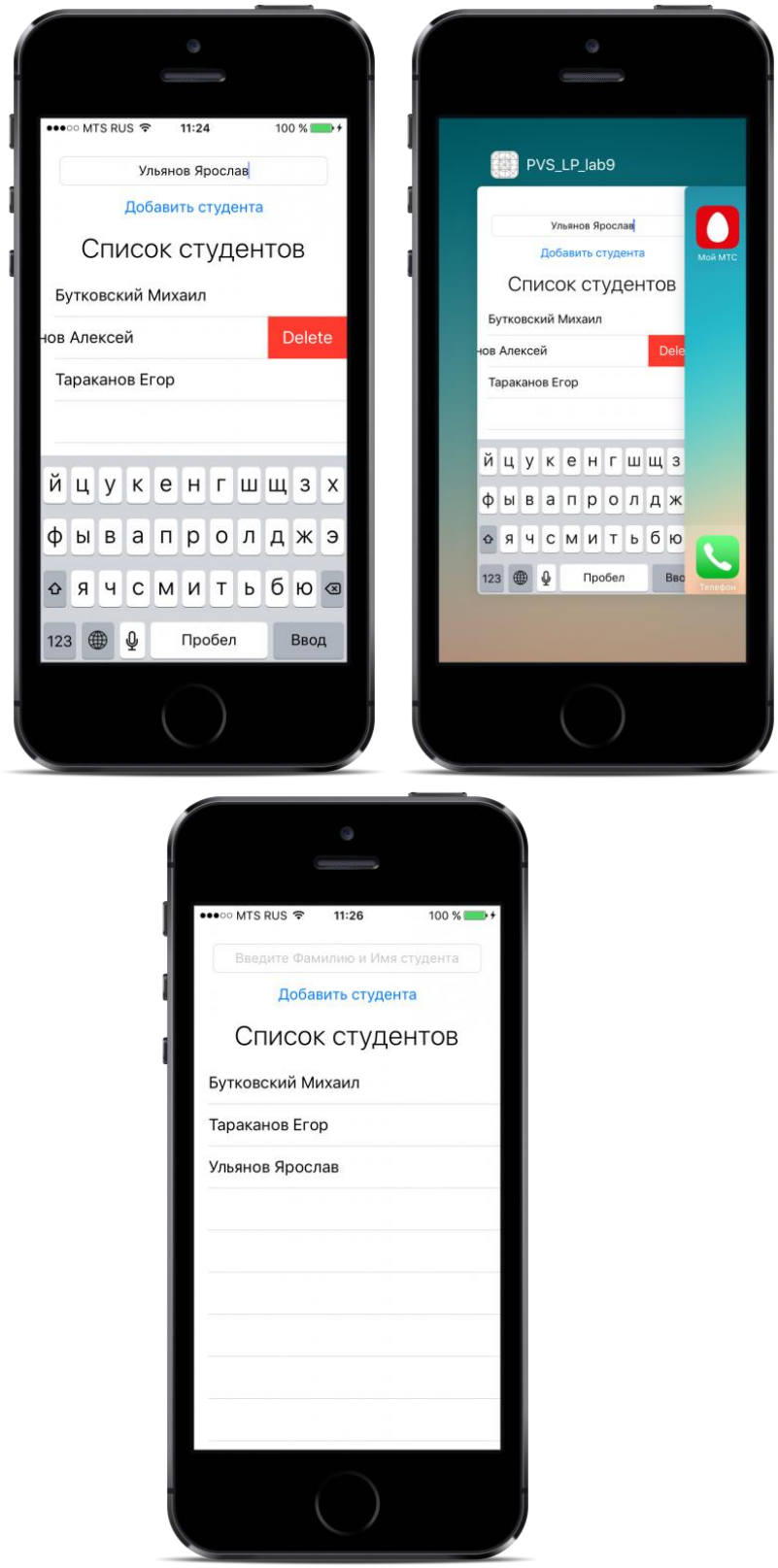


Рис.50. Проверка работы приложения

10.ЛАБОРАТОРНАЯ РАБОТА №10 «СОЗДАНИЕ ПРОСТЕЙШЕГО IOS-КАЛЬКУЛЯТОРА»

Цель работы:

Создать простейший калькулятор для мобильной платформы iOS, используя знания, накопленные в ходе выполнения предыдущих лабораторных работ.

Задание к работе:

Создать калькулятор со следующими возможностями и операциями:

- Сложение;
- Вычитание;
- Умножение;
- Деление;
- Кнопка очистить / очистить все.

Ход работы:

10.1 Создание проекта Xcode.

Создаем проект Xcode, аналогично предыдущим лабораторным работам. Убираем галочку Use Core Data и выбираем iOS – Application – Single View Application.

10.2 Разработка интерфейса приложения.

В область контроллера View добавляем десять кнопок для цифр, пять кнопок для действий, одну кнопку для функции очистки и одно текстовое поле Label для вывода результата (рис.51).

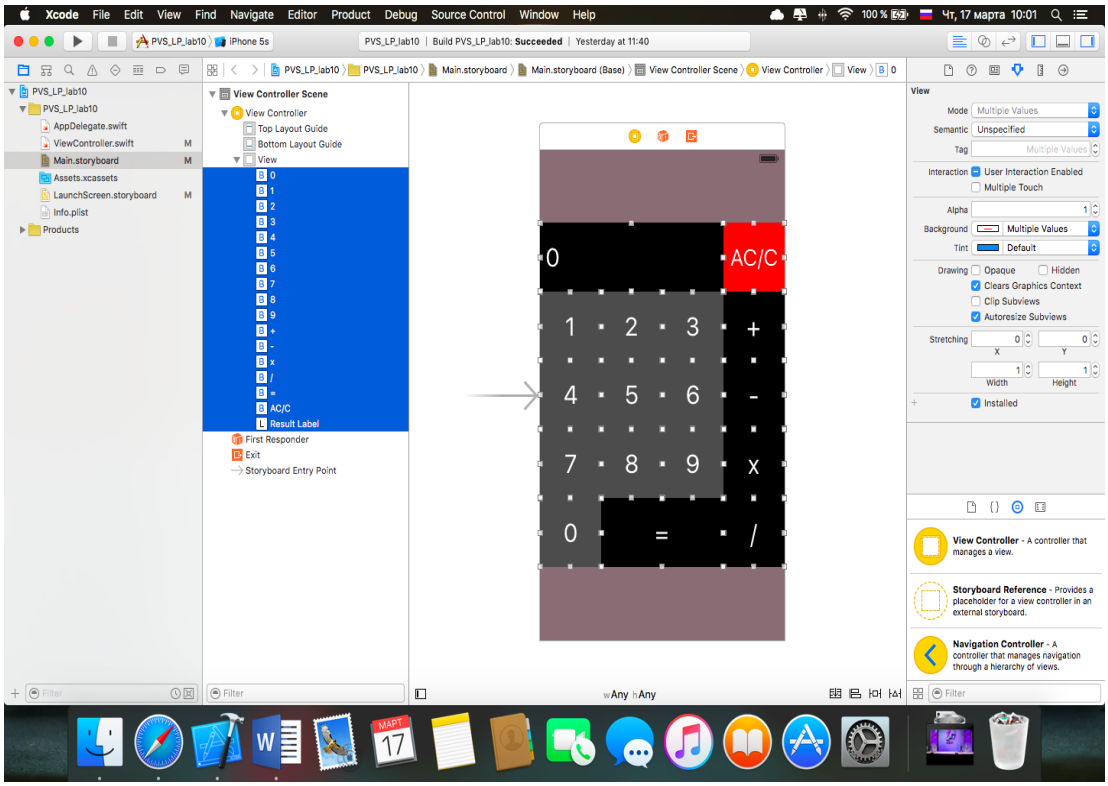


Рис.51. Область контроллера View

В свойствах объекта устанавливаем значение Tag для кнопок цифр (0: “0”, 1: “1”, 2: “2”…9: “9”) и для кнопок действий (+: “1001”, -: ”1002”, x: “1003”, /: “1004” и =: “1005”).

10.3 Создание связей объектов интерфейса с кодом приложения.

Все связи добавляем в описание класса ViewController в файле ViewController.swift.

Для текстового поля Label добавляем связь типа Outlet:

```
@IBOutlet weak var resultLabel: UILabel!
```

Для кнопки любой из цифр добавляем Action-метод:

```
@IBAction func digitsTapped(sender: UIButton)
{
}
```

В тело метода перетаскиваем кнопки оставшихся цифр (рис.52).

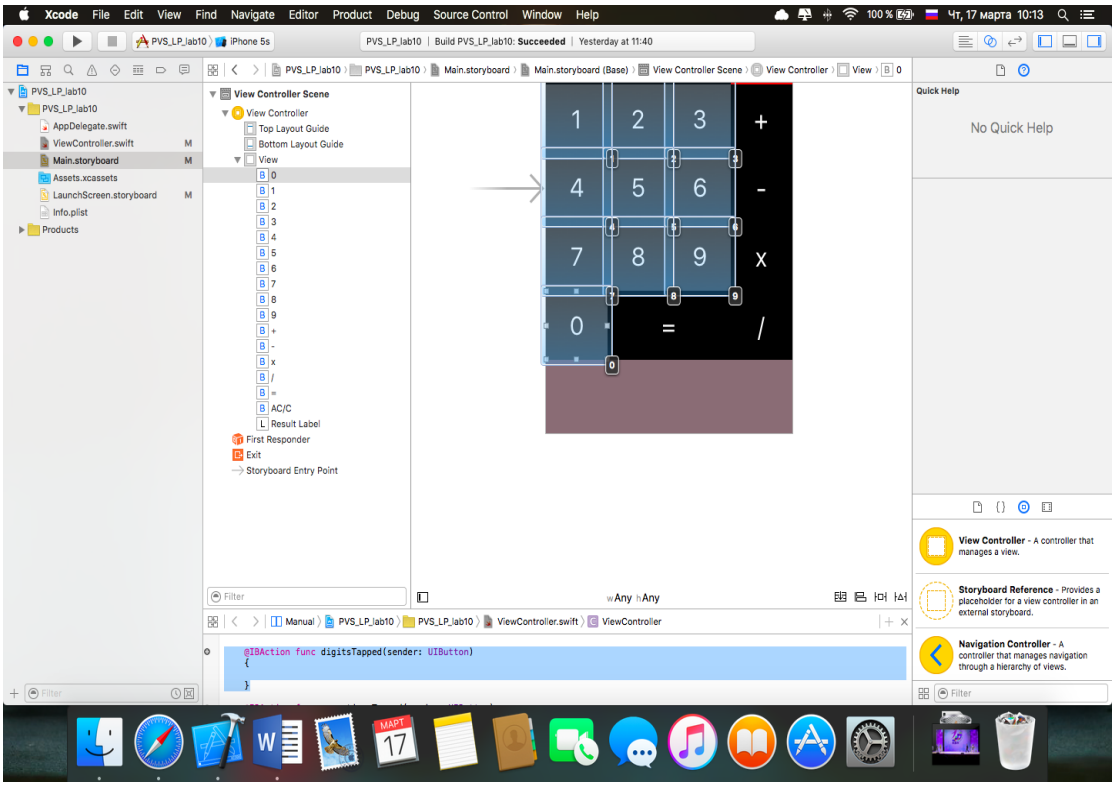


Рис.52. Тело метода

Аналогично, создаем Action-метод для кнопок действий (рис.53):

```
@IBAction func operationsTapped(sender: UIButton)
{

}
```

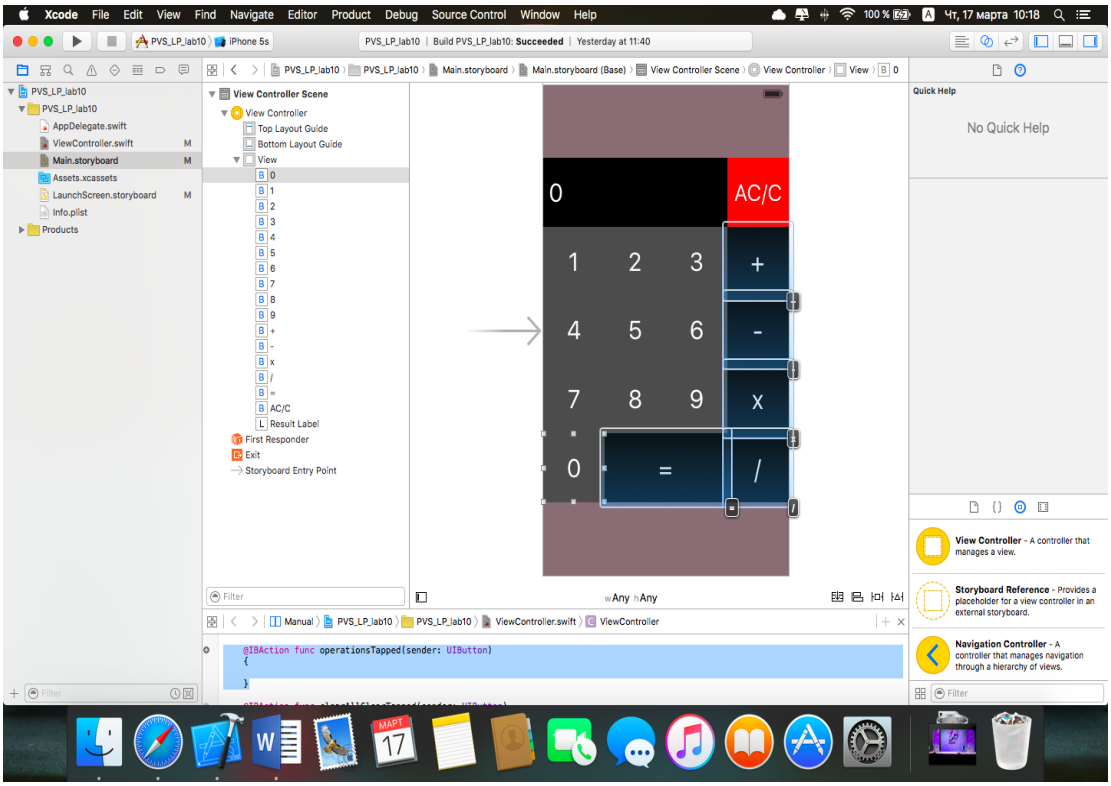


Рис.53. Action-метод для кнопок действий

Для кнопки AC/C добавляем следующий Action-метод:

```
@IBAction func clearAllClearTapped(sender: UIButton)
{

}
```

10.4 Описание событий в коде приложения.

В описание класса ViewController добавляем переменные типа Int для первого и второго числа, равные 0:

```
var x = 0
var y = 0
```

Добавляем переменную типа Int, которая будет отвечать за выбранное действие, теги которых мы указали в свойствах объектов. Также, добавляем переменную типа Bool, с помощью которой мы будем обнулять первое число, по нажатию на какое-либо действие:

```
var enterFlag = false
var operationActive = 0
```

Заполняем Action-метод для кнопок цифр:

```
@IBAction func digitsTapped(sender: UIButton)
{
    if enterFlag == true //Проверяем, было ли нажато какое-либо действие
    {
        x = 0 //Если да, то обнуляем первый аргумент
        enterFlag = false //И присваиваем значение false для флажковой переменной
    }
    x = x * 10 + sender.tag //Значение первого аргумента умножаем на 10 и прибавляем тэг цифры, которая была нажата
    resultLabel.text = " \(x)" //Присваиваем значение первого аргумента текстовому полю с результатом
}
```

Заполняем Ation-метод для кнопок действий:

```
@IBAction func operationsTapped(sender: UIButton)
{
    if operationActive != 0 && enterFlag != true //Если было нажато какое-либо действие и значение флажковой переменной является истиной, то выполняем тело оператора switch
    {
        switch operationActive
        {
        case 1001: //Кнопка +
            x = y + x
```

```
            case 1002: //Кнопка -
                x = y - x
            case 1003: //Кнопка x
                x = y * x
            case 1004: //Кнопка /
                x = y / x
            default: resultLabel.text = " \(x)" //Кнопка =
            }
        }
        operationActive = sender.tag //Переменной, отвечающей за выбранную операцию, присваиваем тэг нажатой операции
        y = x //Значению второго аргумента присваиваем значение первого аргумента
        enterFlag = true //Флажковой переменной присваиваем значение истина
        resultLabel.text = " \(x)" //Присваиваем значение первого аргумента текстовому полю с результатом
    }
```

Заполняем Action-метод для кнопки AC/C:

```
    @IBAction func clearAllClearTapped(sender: UIButton)
    {
        x = 0 //Обнуляем значение первого аргумента
        y = 0 //Обнуляем значение второго аргумента
        resultLabel.text = " 0" //Обнуляем результат в текстовом поле
    }
```

10.5 Сборка проекта и запуск приложения.

Выполняем команду cmd+R и проверяем работу приложения (рис.54).

Рис.54. Проверка работы приложения

Список использованных источников

1. http://www.apple.com/ru/ios/what-is/
2. статья: “Разработчик приложений iOS”, автор: Исследовательский центр портала Superjob.ru
3. веб-сайт: https://developer.apple.com/xcode/features/
4. Айк Харазян “Язык Swift. Самоучитель”, 2016г. 176 стр.

Приложения

Листинг программы лабораторной работы №4.

Файл ViewController.swift

```
import UIKit

class ViewController: UIViewController
{
   @IBOutlet weak var buttonText: UILabel!

   override func viewDidLoad()
   {
      super.viewDidLoad()
   }

   override func didReceiveMemoryWarning()
   {
      super.didReceiveMemoryWarning()
   }

   @IBAction func showHiButton(sender: AnyObject)
   {
      buttonText.text = "Привет!"
   }

   @IBAction func showByeButton(sender: AnyObject)
   {
      buttonText.text = "Пока!"
   }


}
```

Листинг программы лабораторной работы №5.

Файл ViewController.swift

```
import UIKit

class ViewController: UIViewController
{
   @IBOutlet weak var switchIndicator: UILabel!
   @IBOutlet weak var backgroundSwitch: UISwitch!

   override func viewDidLoad()
   {
      super.viewDidLoad()
      switchIndicator.textColor = UIColor.whiteColor()
      switchIndicator.text = "Background image: bg1.jpg"
      view.backgroundColor = UIColor(patternImage: UIImage(named: "bg1")!)
   }

   override func didReceiveMemoryWarning()
   {
      super.didReceiveMemoryWarning()
   }

   @IBAction func backgroundSwitchTapped(sender: AnyObject)
   {
      if backgroundSwitch.on
      {
         switchIndicator.text = "Background image: bg1.jpg"
         view.backgroundColor = UIColor(patternImage: UIImage(named:
"bg1")!)
      }
      else
      {
         switchIndicator.text = "Background image: bg2.jpg"
         view.backgroundColor = UIColor(patternImage: UIImage(named:
"bg2")!)
      }
   }

   }
```

Листинг программы лабораторной работы №6.

Файл ViewController.swift

```
import UIKit

class ViewController: UIViewController
{
   @IBOutlet weak var gestureIndicator: UILabel!

   override func viewDidLoad()
   {
      super.viewDidLoad()
      gestureIndicator.userInteractionEnabled = true
      gestureIndicator.textAlignment = NSTextAlignment.Center
      gestureIndicator.numberOfLines = 2
      gestureIndicator.text = "Используйте жесты в этой области"
      gestureIndicator.backgroundColor = UIColor.yellowColor()
   }

   override func didReceiveMemoryWarning()
   {
      super.didReceiveMemoryWarning()
   }

   @IBAction func tap(sender: AnyObject)
   {
      gestureIndicator.text = "Жест: касание\n Цвет фона: зеленый"
      gestureIndicator.backgroundColor = UIColor.greenColor()
   }

   @IBAction func pinch(sender: AnyObject)
   {
      gestureIndicator.text = "Жест: масштабирование\n Цвет фона: красный"
      gestureIndicator.backgroundColor = UIColor.redColor()
   }
```

```
    @IBAction func rotation(sender: AnyObject)
    {
        gestureIndicator.text = "Жест: вращение\n Цвет фона: синий"
        gestureIndicator.backgroundColor = UIColor.blueColor()
    }

    @IBAction func swipe(sender: AnyObject)
    {
        gestureIndicator.text = "Жест: смахивание\n Цвет фона: серый"
        gestureIndicator.backgroundColor = UIColor.lightGrayColor()
    }

    @IBAction func longPress(sender: AnyObject)
    {
        gestureIndicator.text = "Жест: долгое нажатие\n Цвет фона: оранжевый"
        gestureIndicator.backgroundColor = UIColor.orangeColor()
    }


}
```

Листинг программы лабораторной работы №7.

Файл ViewController.swift

```
import UIKit

class ViewController: UIViewController
{
@IBOutlet weak var SecondNameLabel: UILabel!
@IBOutlet weak var FirstNameLabel: UILabel!
@IBOutlet weak var PatronymicLabel: UILabel!
@IBOutlet weak var AgeLabel: UILabel!
@IBOutlet weak var JobLabel: UILabel!
var SecondName = String()
var FirstName = String()
var Patronymic = String()
var Age = String()
var Job = String()

override func viewDidLoad()
{
super.viewDidLoad()
FirstNameLabel.text = FirstName
SecondNameLabel.text = SecondName
PatronymicLabel.text = Patronymic
AgeLabel.text = Age
JobLabel.text = Job
}

override func didReceiveMemoryWarning()
{
super.didReceiveMemoryWarning()
}



}
```

Файл EditViewController.swift

```
import UIKit

class EditViewController: UIViewController
{
@IBOutlet weak var SecondNameEdit: UITextField!
@IBOutlet weak var FirstNameEdit: UITextField!
@IBOutlet weak var PatronymicEdit: UITextField!
@IBOutlet weak var AgeEdit: UITextField!
@IBOutlet weak var JobEdit: UITextField!

override func viewDidLoad()
{
super.viewDidLoad()
}

override func didReceiveMemoryWarning()
{
super.didReceiveMemoryWarning()
}

override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?)
{
let DestinationViewController : ViewController = segue.destinationViewController as! ViewController
DestinationViewController.FirstName = FirstNameEdit.text!
DestinationViewController.SecondName = SecondNameEdit.text!
DestinationViewController.Patronymic = PatronymicEdit.text!
DestinationViewController.Age = AgeEdit.text!
DestinationViewController.Job = JobEdit.text!
}


}
```

Листинг программы лабораторной работы №8.

Файл StudentsTableViewController.swift

```
import UIKit

class StudentsTableViewController: UITableViewController
{
   var students = [String]()
   var newStudent:String = ""

   override func viewDidLoad()
   {
      super.viewDidLoad()
   }

   override func didReceiveMemoryWarning()
   {
      super.didReceiveMemoryWarning()
   }

   override func numberOfSectionsInTableView(tableView: UITableView) ->
Int
   {
      return 1
   }

   override func tableView(tableView: UITableView, numberOfRowsInSection
section: Int) -> Int
   {
      return students.count
   }

   override func tableView(tableView: UITableView, cellForRowAtIndexPath
indexPath: NSIndexPath) -> UITableViewCell
   {
```

```
        let cell = tableView.dequeueReusableCellWithIdentifier("studentCell", for-
IndexPath: indexPath)
        cell.textLabel!.text = students[indexPath.row]
        return cell
    }

    override func tableView(tableView: UITableView, commitEditingStyle edit-
ingStyle: UITableViewCellEditingStyle, forRowAtIndexPath indexPath: NSIndex-
Path)
    {
        if editingStyle == UITableViewCellEditingStyle.Delete
        {
            students.removeAtIndex(indexPath.row)
            tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation:
UITableViewRowAnimation.Left)
        }
    }

    @IBAction func done(segue: UIStoryboardSegue)
    {
        let addStudentNameVC = segue.sourceViewController as! AddStu-
dentViewController
        newStudent = addStudentNameVC.studentNameToAdd
        students.append(newStudent)
        self.tableView.reloadData()
    }

    @IBAction func cancel(segue: UIStoryboardSegue)
    {

    }


}
```

Файл AddStudentViewController.swift

```
import UIKit

class AddStudentViewController: UIViewController
{
   @IBOutlet weak var studentNameTextField: UITextField!
   var studentNameToAdd:String = ""

   override func viewDidLoad()
   {
      super.viewDidLoad()
   }

   override func didReceiveMemoryWarning()
   {
      super.didReceiveMemoryWarning()
   }

   override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyOb-
ject?)
   {
      if segue.identifier == "doneSegue"
      {
         studentNameToAdd = studentNameTextField.text!
      }
   }


}
```

Листинг программы лабораторной работы №9

Файл ViewController.swift

```
import UIKit
import CoreData

class ViewController: UIViewController, UITableViewDataSource
{
   @IBOutlet weak var tableView: UITableView!
   @IBOutlet weak var studentNameTextField: UITextField!

   Var students = [NSManagedObject]()

   override func viewDidLoad()
   {
      super.viewDidLoad()
   }

   override func didReceiveMemoryWarning()
   {
      super.didReceiveMemoryWarning()
   }

   func numberOfSectionsInTableView(tableView: UITableView) -> Int
   {
      return 1
   }

   func tableView(tableView: UITableView, numberOfRowsInSection section:
Int) -> Int
   {
      return students.count
   }

   func tableView(tableView: UITableView, cellForRowAtIndexPath index-
Path: NSIndexPath) -> UITableViewCell
```

```
        {
            let cell = tableView.dequeueReusableCellWithIdentifier(«studentCell»)!
As UITableViewCell

            let student = students[indexPath.row]
            cell.textLabel?.text = student.valueForKey(«name») as? String

            return cell
        }

        func tableView(tableView: UITableView, commitEditingStyle editingStyle:
UITableViewCellEditingStyle, forRowAtIndexPath indexPath: NSIndexPath)
        {
            if editingStyle == UITableViewCellEditingStyle.Delete
            {
                let appDelegate = UIApplication.sharedApplication().delegate as! Ap-
pDelegate
                let managedObjectContext = appDelegate.managedObjectContext

                managedObjectContext.deleteObject(students[indexPath.row] as
NSManagedObject)
                do
                {
                    try managedObjectContext.save()
                    students.removeAtIndex(indexPath.row)
                    tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation:
.Left)
                }
                catch let error as NSError
                {
                    print(«Data removing error: \(error)»)
                }
            }
        }

        @IBAction func addStudentButton(sender: AnyObject)
        {
```

```
        if studentNameTextField.text == «» || studentNameTextField.text == «Введите данные!»
        {
            studentNameTextField.text = «Введите данные!»
        }
        else
        {
            let appDelegate = UIApplication.sharedApplication().delegate as! AppDelegate
            let managedObjectContext = appDelegate.managedObjectContext

            let newObject = NSEntityDescription.insertNewObjectForEntityForName(«Students», inManagedObjectContext: managedObjectContext) as NSManagedObject

            newObject.setValue(studentNameTextField.text!, forKey: «name»)

            do
            {
                try managedObjectContext.save()
                students.append(newObject)
                studentNameTextField.text! = «»
                self.tableView.reloadData()
                self.view.endEditing(true)
            }
            catch let error as NSError
            {
                print(«Data saving error: \(error)»)
            }
        }
    }

    override func viewWillAppear(animated: Bool)
    {
        super.viewWillAppear(animated)
```

```
            let appDelegate = UIApplication.sharedApplication().delegate as! AppDel-
egate
            let managedObjectContext = appDelegate.managedObjectContext

            let fetchRequest = NSFetchRequest(entityName: «Students»)

            do
            {
               students = try managedObjectCon-
text.executeFetchRequest(fetchRequest) as! [NSManagedObject]
            }
            catch let error as NSError
            {
               print(«Data loading error: \(error)»)
            }
            self.tableView.reloadData()
         }


      }
```

Листинг программы лабораторной работы №10.

Файл ViewController.swift

```
import UIKit

class ViewController: UIViewController
{
   @IBOutlet weak var resultLabel: UILabel!

   var x = 0
   var y = 0

   var enterFlag = false
   var operationActive = 0

   override func viewDidLoad()
   {
      super.viewDidLoad()
   }

   override func didReceiveMemoryWarning()
   {
      super.didReceiveMemoryWarning()
   }

   @IBAction func digitsTapped(sender: UIButton)
   {
      if enterFlag == true
      {
         x = 0
         enterFlag = false
      }
      x = x * 10 + sender.tag
      resultLabel.text = " \(x)"
   }
```

```
    @IBAction func operationsTapped(sender: UIButton)
    {
        if operationActive != 0 && enterFlag != true
        {
            switch operationActive
            {
            case 1001:
                x = y + x
            case 1002:
                x = y - x
            case 1003:
                x = y * x
            case 1004:
                x = y / x
            default: resultLabel.text = " \(x)"
            }
        }
        operationActive = sender.tag
        y = x
        enterFlag = true
        resultLabel.text = " \(x)"
    }

    @IBAction func clearAllClearTapped(sender: UIButton)
    {
        x = 0
        y = 0
        resultLabel.text = " 0"
    }


}.
```