ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ ДЛЯ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

Лабораторная работа № 9. Разработка приложений на языке Swift и Objective-C: геолокация, Mapkit и CoreData

Содержание

ВВЕДЕНИЕ

Данные методические указания предназначены для выполнения лабораторных работ по дисциплине «Технологии программирования» студентами 2 курса специальности «Прикладная информатика».

СТРУКТУРА ЛАБОРАТОРНОЙ РАБОТЫ

Каждая лабораторная работа содержит тексты задач и контрольные вопросы, ответы на которые проверяются преподавателем при приеме работы у студента. Для некоторых задач приводятся указания по их решению.

Каждая задача имеет уникальный в пределах данной лабораторной работы номер. В ссылках на задачу указывается номер задачи и в скобках номер страницы.

В приложение вынесены требования и рекомендации по оформлению исходных текстов программ.

ПОРЯДОК СДАЧИ ЛАБОРАТОРНОЙ РАБОТЫ

Выполнение студентом лабораторной работы и сдача ее результатов преподавателю происходит следующим образом.

1. Студент выполняет разработку программ.

   В ходе разработки студент обязан следовать указаниям к данной задаче (в случае их наличия). Исходные тексты программ следует разрабатывать в соответствии с требованиями к оформлению, приведенными в приложении.

2. Студент выполняет самостоятельную проверку исходного текста каждой разработанной программы и правильности ее работы, а также свои знания по теме лабораторной работы.

   Исходные тексты программ должны соответствовать требованиям к оформлению, приведенным в приложении. Недопустимо отсутствие в тексте программы следующих важных элементов оформления: спецификации программного файла и подпрограмм, а также отступов, показывающих структурную вложенность языковых конструкций.

   Для проверки правильности работы программы студенту необходимо разработать набор тестов и на их основе провести тестирование программы. Тестовый набор должен включать примеры входных и выходных данных, приведенные в тексте задачи, а также тесты, разработанные студентом самостоятельно.

   Самостоятельная проверка знаний по теме лабораторной работы выполняется с помощью контрольных вопросов и заданий, приведенных в конце текста лабораторной работы.

3. Студент защищает разработанные программы. Защита заключается в том, что студент должен ответить на вопросы преподавателя, касающиеся разработанной программы, и контрольные вопросы.

К защите необходимо представить исходные тексты программ, оформленных в соответствии с требованиями, и протоколы тестирования каждой программы, подтверждающие правильность ее работы.

Протокол тестирования включает в себя тест (описание входных данных и соответствующих им выходных данных), описание выходных данных, полученных при запуске программы на данном тесте, и отметку о прохождении теста. Тест считается пройденным, если действительные результаты работы программы совпали с ожидаемыми.

ТРЕБОВАНИЯ К ОТЧЁТУ И ОФОРМЛЕНИЮ КОДА

В файле `Readme` проекта на `github` должна быть ссылка на отчёт. Отчет опубликовать в репозитории в папке `docs` или во внешнем хранилище, добавив ссылку на него в файл `Readme` репозитория.

Отчет по лабораторной работе содержит тексты задач, фрагменты кода, описывающие основную функциональность и скриншот разработанного приложения в симуляторе.

Код проекта должен содержать комментарии в стиле Markdown, комментарии PRAGMA, справочные комментарии (см. лабораторную работу 7) и сведения об авторе.

Исходный код приложений должен быть оформлен согласно руководству стиля в зависимости от языка:

1. Swift:
   a) http://ilya2606.ru/?p=1846
   b) https://github.com/RedMadRobot/RMR-swift-style-guide
   c) https://github.com/raywenderlich/swift-style-guide
2. Objective-C:
   a) https://github.com/DigDes/objective-c-style-guide
   b) https://github.com/raywenderlich/objective-c-style-guide

Репозиторий должен содержать следующие ветки:

1. **main** — файл Readme со ссылкой на отчет и описание веток и выполненных заданий.
2. **example-task1** — каталог проекта example1, содержащий проект для примера 1.1
3. **example-task2** — каталог проекта example2, содержащий проект для примера 1.2

4. **example-task3** — каталог проекта example3, содержащий проект для примера 1.3
5. **example-task4** — каталог проекта example4, содержащий проект для примера 1.4
6. **example-task5** — каталог проекта example5, содержащий проект для примера 1.5
7. **example-task6** — каталог проекта example6, содержащий проект для примера 1.6
8. **example-task7** — каталог проекта example7, содержащий проект для примера 1.7
9. **feature-task2-1** — каталог проекта task2-1, содержащий проект для задачи 2.1.
10. **feature-task2-2** — каталог проекта task2-2, содержащий проект для задачи 2.2.

КРИТЕРИИ ОЦЕНИВАНИЯ

Оценка 4

Выполнены **все примеры для изучения**. Отчёт содержит описание задачи, ключевые фрагменты кода и скриншот приложения в симуляторе или на телефоне. Отчёт должен быть опубликован в репозитории или должна быть ссылка на него в файле `Readme`. Отчёт и исходный код проекта может содержать ошибки. Максимальная задержка на предоставление отчёта, исходного кода проекта, включая публикацию в репозитории и задании курса, а также его защиту — 2 недели.

Оценка 5

Выполнены **все примеры для изучения** и даны **ответы на контрольные вопросы**. Отчёт содержит описание задачи, ключевые фрагменты кода, скриншот приложения в симуляторе или на телефоне и ответы на контрольные вопросы. Отчёт должен быть опубликован в репозитории или должна быть ссылка на него в файле `Readme`. Отчёт и исходный код проекта может содержать ошибки. Максимальная задержка на предоставление отчёта, исходного кода проекта, включая публикацию в репозитории и задании курса, а также его защиту — 2 недели.

Оценка 6-7

Выполнены **примеры для изучения** и реализовано **приложение согласно задания 2.1** и даны **ответы на контрольные вопросы**. Отчёт содержит описание задачи, ключевые фрагменты кода, скриншот приложения в симуляторе или на телефоне и ответы на контрольные вопросы. Отчёт должен быть опубликован в репозитории или должна быть ссылка на него в файле `Readme`. Отчёт и исходный код проекта может содержать ошибки. Максимальная задержка на предоставление отчёта, исходного кода проекта, включая публикацию в репозитории и задании курса, а также его защиту — 1 неделя.

Оценка 8

Рассмотрены **примеры для изучения**, реализованы **приложения согласно заданиям 2.1-2.2** и даны **ответы на контрольные вопросы**. Отчёт содержит описание задачи, ключевые фрагменты кода, скриншот приложения в симуляторе или на телефоне и ответы на контрольные вопросы. Отчёт должен быть опубликован в репозитории или должна быть ссылка на него в файле `Readme`. Отчёт и исходный код могут содержать незначительные ошибки. Максимальная задержка на предоставление отчёта, исходного кода проекта, включая публикацию в репозитории и задании курса, а также его защиту — 1 неделя.

Оценка 9

Рассмотрены **примеры для изучения**, реализованы **приложения согласно заданиям 2.1-2.2** и даны **ответы на контрольные вопросы**. Отчёт содержит описание задачи, ключевые фрагменты кода, скриншот приложения в симуляторе или на телефоне и ответы на контрольные вопросы. Отчёт должен быть опубликован в репозитории или должна быть ссылка на него в файле `Readme`. Лабораторная работа опубликована в репозитории и в задании курса в срок, а так же защищена в срок. Не содержит ошибок.

1. ПРИМЕРЫ ДЛЯ ИЗУЧЕНИЯ

1.1. iOS ПРИЛОЖЕНИЕ С ХРАНЕНИЕМ ДАННЫХ В PLIST НА ЯЗЫКЕ SWIFT

Цель: изучить примеры использования plist.

Просмотреть видео и изучить примеры из видео:

- https://www.youtube.com/watch?v=Xn0ZO5eXSvY
- https://www.youtube.com/watch?v=Yb4n2DpEiTA
- https://www.youtube.com/watch?v=0nxRZXg6KQY

Реализовать приложение из примеров.

1.2. ПРИЛОЖЕНИЕ С АВТОРИЗАЦИЕЙ И ХРАНЕНИЕМ ДАННЫХ В NSUSERDEFAULTS НА ЯЗЫКЕ SWIFT

ЗАДАЧИ И МЕТОДИЧЕСКИЕ УКАЗАНИЯ

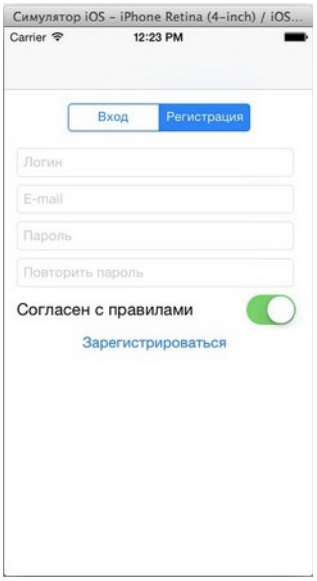
Требуется разработать приложение с формой авторизации и хранением данных пользователей в NSUserDefaults

Входные данные

Форма авторизации / регистрации

Выходные данные

Уведомление об успешной авторизации или регистрации



В примере создадим форму для логина и регистрации (см. рис. выше), изучим функцию **Segmented Control**, а также разберем, что такое **NSUserDefaults**.

```
- (IBAction)segment:(id)sender {
    UISegmentedControl *seg = (UISegmentedControl *) sender;
    NSInteger select = seg.selectedSegmentIndex;

    if(select ==0){
        [_regV setHidden:YES];
    }
    else {
        [_regV setHidden:NO];
    }

}
- (IBAction)switch:(id)sender {
}
- (IBAction)buttonL:(id)sender {
    if(![_loginL.text isEqualToString:@""] || ![_passL.text isEqualToString:@""]){
    NSUserDefaults *user = [NSUserDefaults standardUserDefaults];
    [user setObject:_loginL.text forKey:@"login"];
    [user setObject:_passL.text forKey:@"pass"];
    [self performSegueWithIdentifier:@"closeZone" sender:self];
    }
}
```

Итак, сначала мы создадим форму для логина, которая будет состоять из поля для логина пользователя, его пароля, а также кнопки «Вход». Далее мы создадим функционал **Segment Control**, который будет переключать нас между вкладками «Логин» и «Регистрация». Чтобы сделать форму для регистрации, сначала необходимо создать объект **UIView** внутри **ViewController**, а уже затем будем добавлять в него **TextField** для регистрации. Также добавим в форму регистрации новый элемент — **UISwitch**, это стандартный переключатель, который может принимать значения либо ON, либо OFF. Этот переключатель нам понадобится, чтобы пользователь мог согласиться или не согласиться с предлагаемыми правилами приложения.

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    [_regV setHidden:YES];
    NSUserDefaults *user = [NSUserDefaults standardUserDefaults];

    if([user objectForKey:@"login"] !=nil || [user objectForKey:@"pass"]!=nil){
        [self performSegueWithIdentifier:@"closeZone" sender:self];

    }
    // Do any additional setup after loading the view, typically from a nib.
}
```

Переходим к созданию функционала для функции входа. После нажатии на кнопку **Логин** будет срабатывать функция, которая проверяет, заполнены ли все поля, если да, то значения из этих полей будут занесены в кэш программы — **NSUserDefaults**. А пользователь попадет в закрутую часть приложения. Стоит отметить, что теперь, после запуска программы пользователь будет попадать в закрытую зону, пока не выйдет из аккаунта или программа будет удалена.

Дополнительные материалы

http://macbug.ru/cocoa/userdefns

https://adobkin.com/2015/06/16/sokhranieniie-polzovatielskikh-nastroiek-s-pomoshchiu-nsuserdefaults-chast-1/

http://www.securitylab.ru/analytics/450323.php

1.3. iOS ПРИЛОЖЕНИЕ С ФУНКЦИЕЙ ГЕОЛОКАЦИИ НА ЯЗЫКЕ SWIFT

Цель: изучить примеры определения местоположения пользователя с использованием библиотеки CoreLocation

1. Просмотреть видео: https://www.youtube.com/watch?v=eFf4MWBbEZQ
2. Изучить статью и документацию:
    - http://www.seemuapps.com/swift-get-users-location-gps-coordinates
    - https://developer.apple.com/documentation/corelocation/getting_the_user_s_location

Реализовать приложение из примеров.

1.4. iOS ПРИЛОЖЕНИЕ С ФУНКЦИЕЙ ОПРЕДЕЛЕНИЯ МЕСТОПОЛОЖЕНИЯ ПО КАРТЕ НА ЯЗЫКЕ SWIFT

Цель: изучить примеры определения местоположения пользователя с использованием библиотек MapKit и CoreLocation

Изучить статью и документацию:

- https://www.appcoda.com/geo-targeting-ios/
- https://www.spaceotechnologies.com/current-gps-location-ios-app-core-location-framework/

- https://developer.apple.com/documentation/corelocation/getting_the_user_s_location

Реализовать приложение из примеров.

1.5. СИСТЕМА БРОНИРОВАНИЯ БИЛЕТОВ НА OBJECTIVE-C

ЗАДАЧИ И МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Изучить пример и разработать программу бронирования авиабилетов, используя CoreData и MapKit. Пользователь выбирает маршрут на карте, и ему предлагается список рейсов, включающих название авиакомпании, стоимость перелёта. Данные должны храниться в базе данных sqlite, доступ через CoreData.

Входные данные

Маршрут, состоящий из набора городов.

Выходные данные

Список рейсов, включая время и стоимость.

Создайте новый Single View Application проект. Добавьте в интерфейс приложения компонент MKMapView (Рис. 1). Создайте IBOutlet карты в контроллере. Удерживая нажатой клавишу Ctrl перетащите MKMapView в область interface файла ViewController.m используя режим Assistant editor (Рис. 2). Введите map и нажмите Connect (Рис. 3).

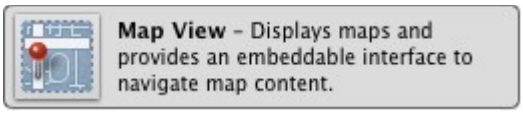


Рис. 1. Компонент MKMapView

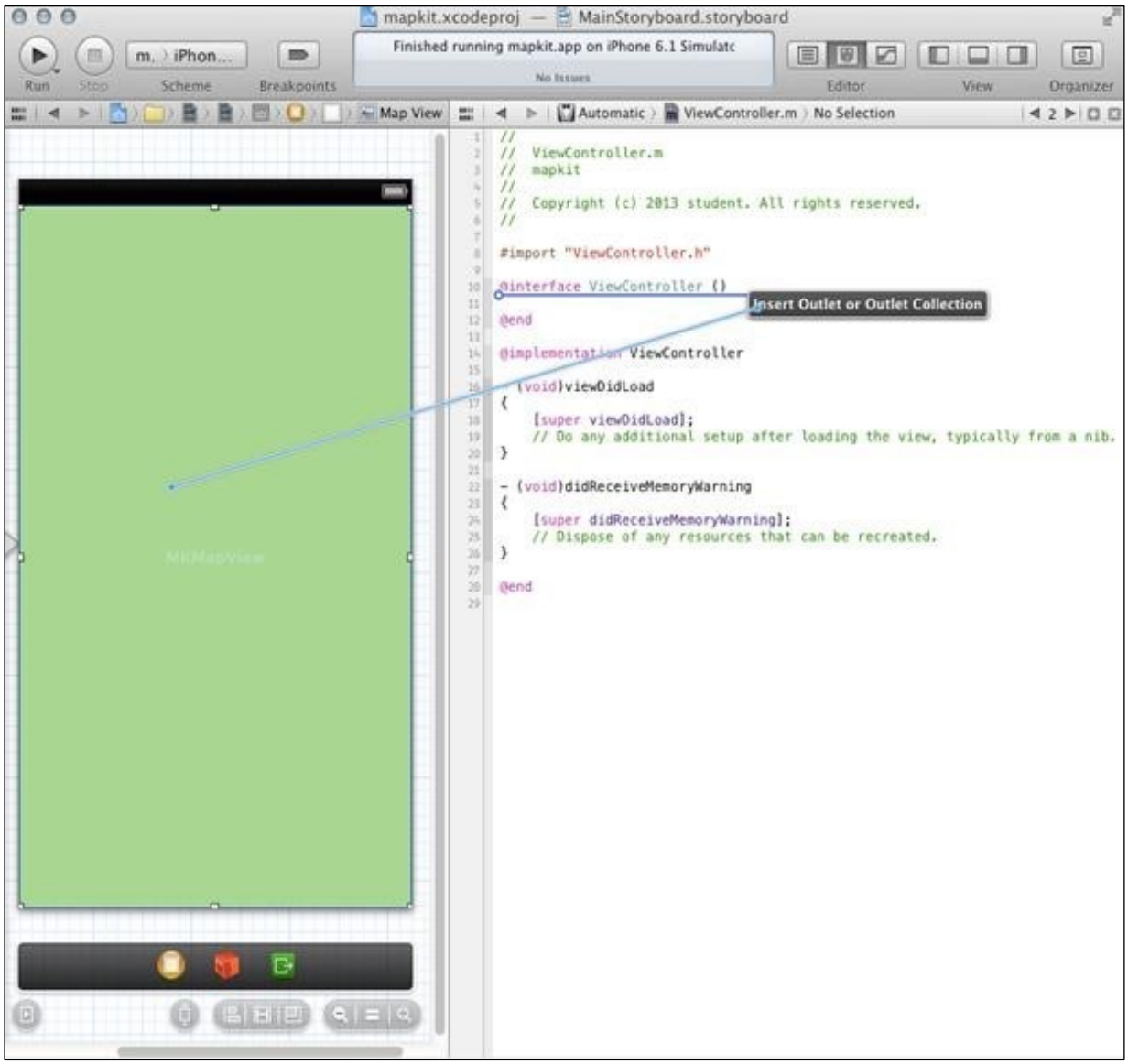


Рис. 2. Создание IBOutlet

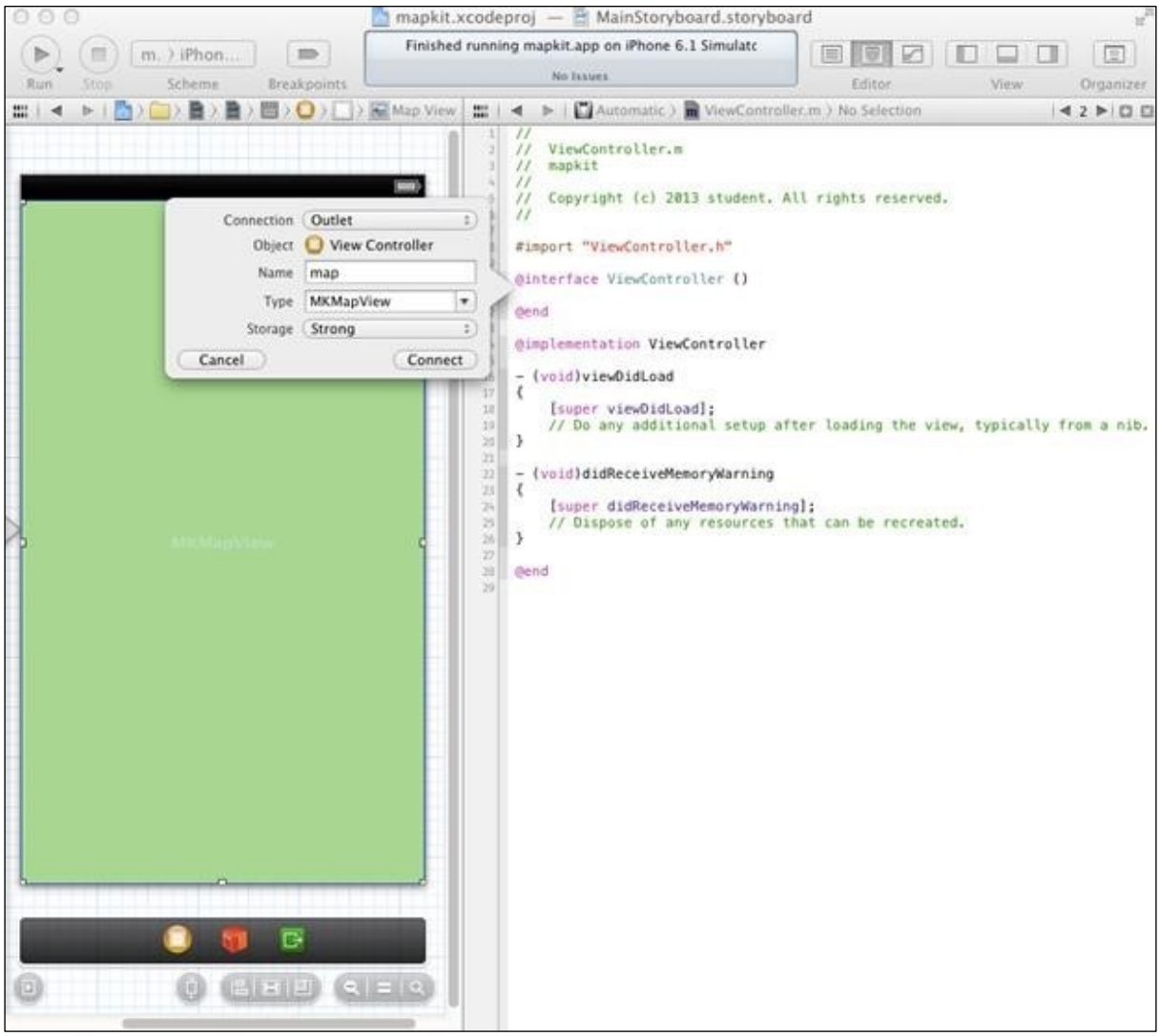


Рис. 3. Создание связи

В область interface добавиться свойство карты (Листинг 1).

```
@property (weak, nonatomic) IBOutlet MKMapView *map;
```

Листинг 1. Свойство карты

При попытке запустить приложение появится ошибка. Необходимо добавить фреймворк MapKit (Рис. 4). Для этого выберите файл проекта в навигаторе.

Рис. 4. Компонент MapKit

Выберите Target – mapkit. Затем перейдите во вкладку Build Phases. Раскройте пункт Link Binary With Libraries, нажмите ‘+’, введите map и выберите MapKit.framework (Рис. 5).

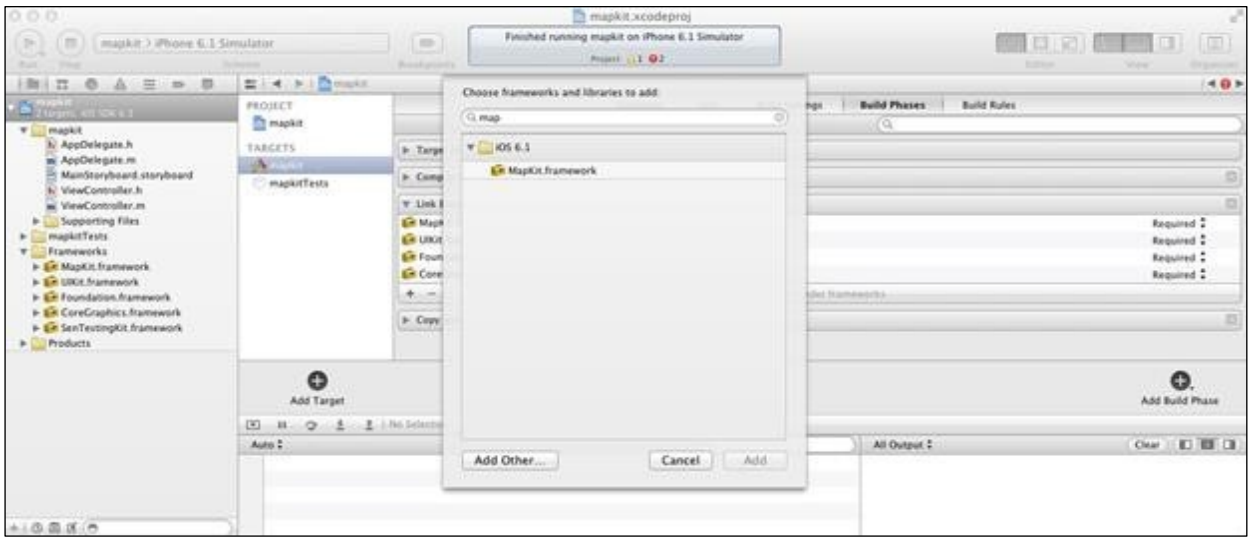


Рис. 5. Добавление фреймворка

Создание базы данных Core Data. Открываем проект, в нём нажимаем Ctrl+N и выбираем Core Data → Data Model (Рис. 6).

Рис. 6. Создание проекта Core Data

Создадим новую сущность (Рис. 7).

Рис. 7. Кнопка добавления сущности

Переименуем сущность, назвав ее Record (Рис. 8).

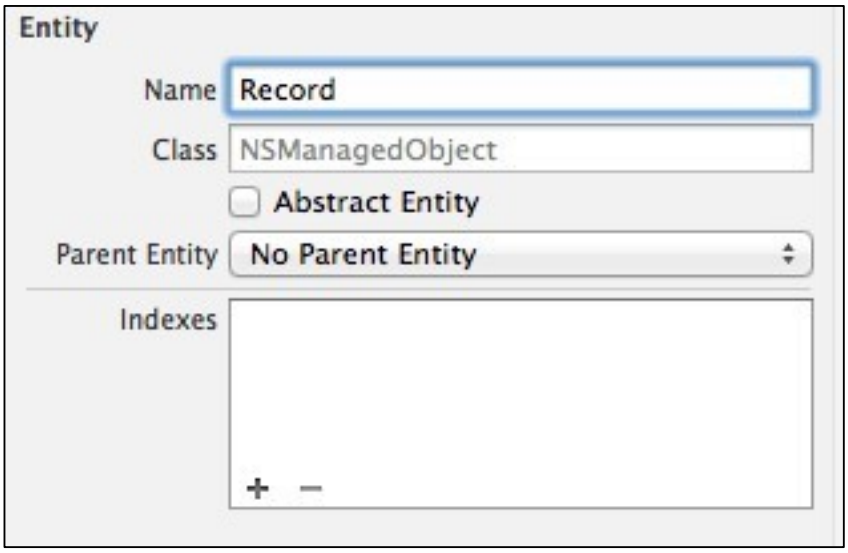


Рис. 8. Создание сущности

Добавим 4 свойства: aviaCompany (String), cityFrom (String), cityTo (String), price (Float) (Рис. 9).

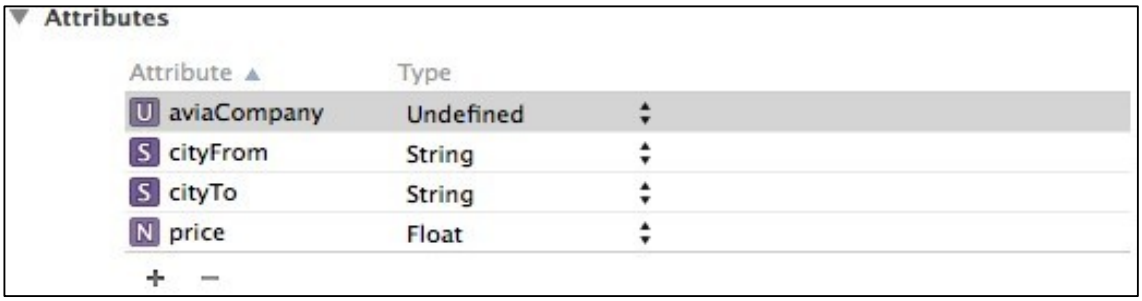


Рис. 9. Добавление свойств

Далее необходимо добавить фреймворк CoreData. Для этого выберите файл проекта в навигаторе (Рис. 10).

Рис. 10. Файл проекта

Выберите на вкладке свойств Target – mapkit. Затем перейдите во вкладку Build Phases. Раскройте пункт Link Binary With Libraries, нажмите ‘+’, введите CoreData и выберите CoreData.framework (Рис. 11).

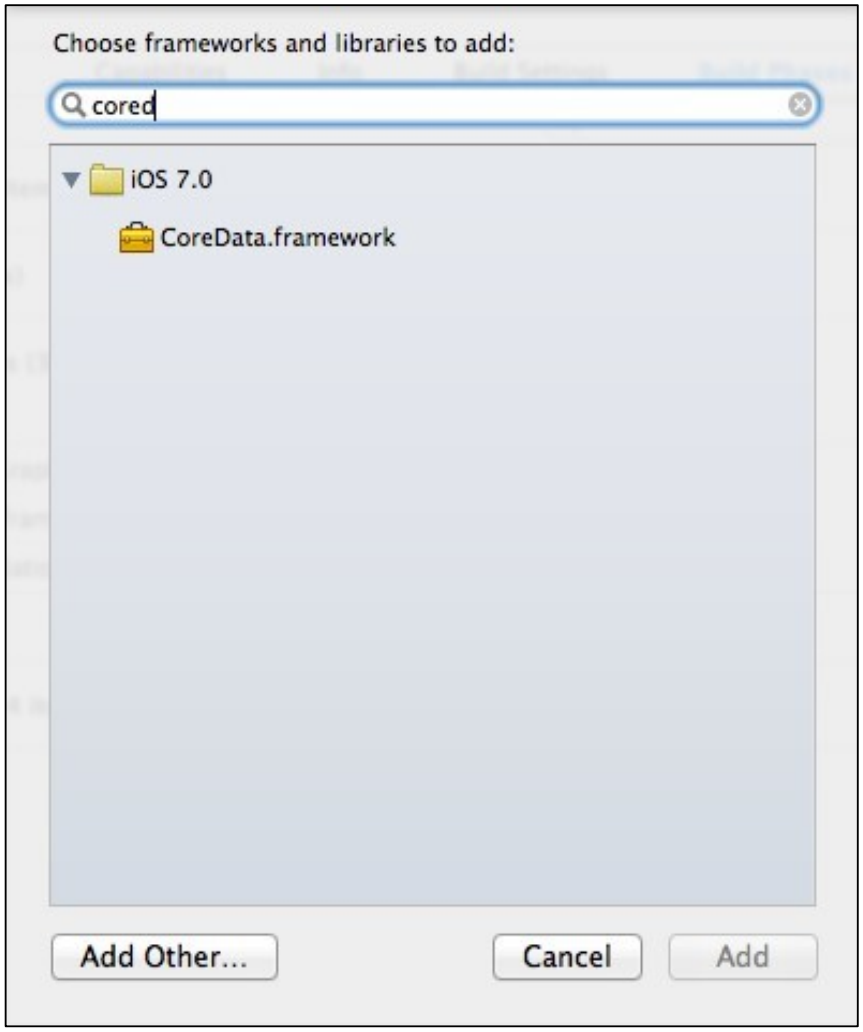


Рис. 11. Добаление фреймворка

Откройте файл AppDelegate.h и подключите фреймворк (Листинг 2).

```
#import <CoreData/CoreData.h>
```

Листинг 2. Подключение фреймворка

Также добавьте в файл AppDelegate.h следующие свойства для использования базы данных (Листинг 3)

```
@property (nonatomic, strong) NSManagedObjectModel *managedObjectModel;

@property (nonatomic, strong) NSPersistentStoreCoordinator
*persistentStoreCoordinator;

@property (nonatomic, strong) NSManagedObjectContext *managedObjectContext;
```

Листинг 3. Добавление свойств

Откройте файл AppDelegate.m и добавьте в @implementation описание сущностей (Листинг 4).

```
@synthesize
managedObjectContext,managedObjectModel,persistentStoreCoordinator;
```

Листинг 4. Описание сущностей

Там же необходимо добавить методы инициализации (Листинг 5).

```
// 1
- (NSManagedObjectContext *)
managedObjectContext {
    if (managedObjectContext != nil) {
        return managedObjectContext;
    }

    NSPersistentStoreCoordinator *coordinator =
[self persistentStoreCoordinator];
    if (coordinator != nil) {
        managedObjectContext =
[[NSManagedObjectContext alloc] init];
        [managedObjectContext
setPersistentStoreCoordinator: coordinator];
    }

    return managedObjectContext;
}
//2
 - (NSManagedObjectModel *)managedObjectModel {
    if (managedObjectModel != nil) {
        return managedObjectModel;
    }

      managedObjectModel = [NSManagedObjectModel mergedModelFromBundles:nil];

          return managedObjectModel;
      }

      - (NSURL *)applicationDocumentsDirectory {
          return [[[NSFileManager defaultManager]
URLsForDirectory:NSDocumentDirectory inDomains:NSUserDomainMask] lastObject];
      }
```

```
//3
- (NSPersistentStoreCoordinator *)persistentStoreCoordinator {
    if (persistentStoreCoordinator != nil) {
        return persistentStoreCoordinator;
    }

    NSURL *storeURL = [[self applicationDocumentsDirectory]
URLByAppendingPathComponent:@"flight.sqlite"];

    NSError *error = nil;
    persistentStoreCoordinator = [[NSPersistentStoreCoordinator
alloc] initWithManagedObjectModel:[self managedObjectModel]];

    if (![persistentStoreCoordinator
addPersistentStoreWithType:NSSQLiteStoreType configuration:nil
URL:storeURL options:nil error:&error]) {

    }

    return persistentStoreCoordinator;
}
```

Листинг 5. Методы инициализации

Первый, второй и третий методы будут возвращать объекты "managedObjectContext", "managedObjectModel" и "persistentStoreCoordinator" соответственно, связанные с базой данных из файла (Sqlite). Далее нам необходимо генерировать класс для работы с базой данных.

Создаем каталог Model и выбираем его как текущий. Нажимаем Ctrl+N, выбираем Core Data → NSManageObjectSubclass (Рис. 12).

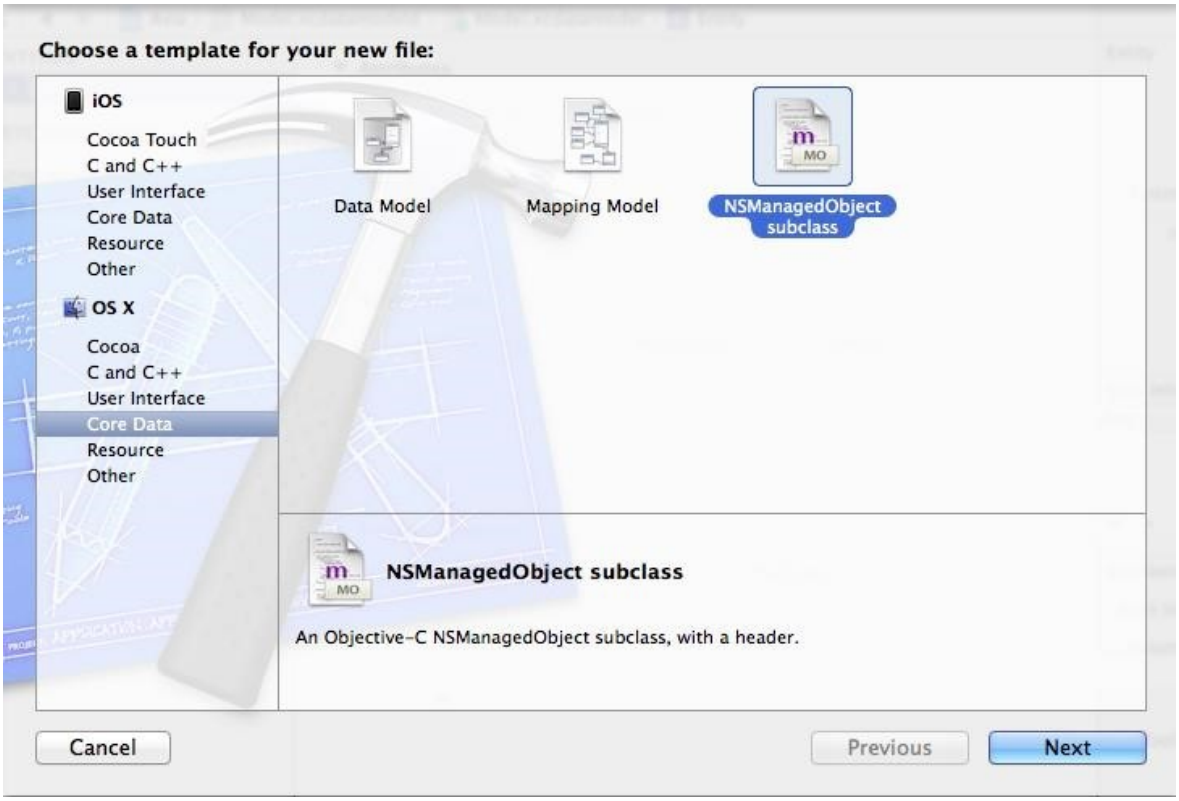


Рис. 12. Создание управляемого объекта

Выберите ранее созданную модель и нажмите Next (Рис. 13).

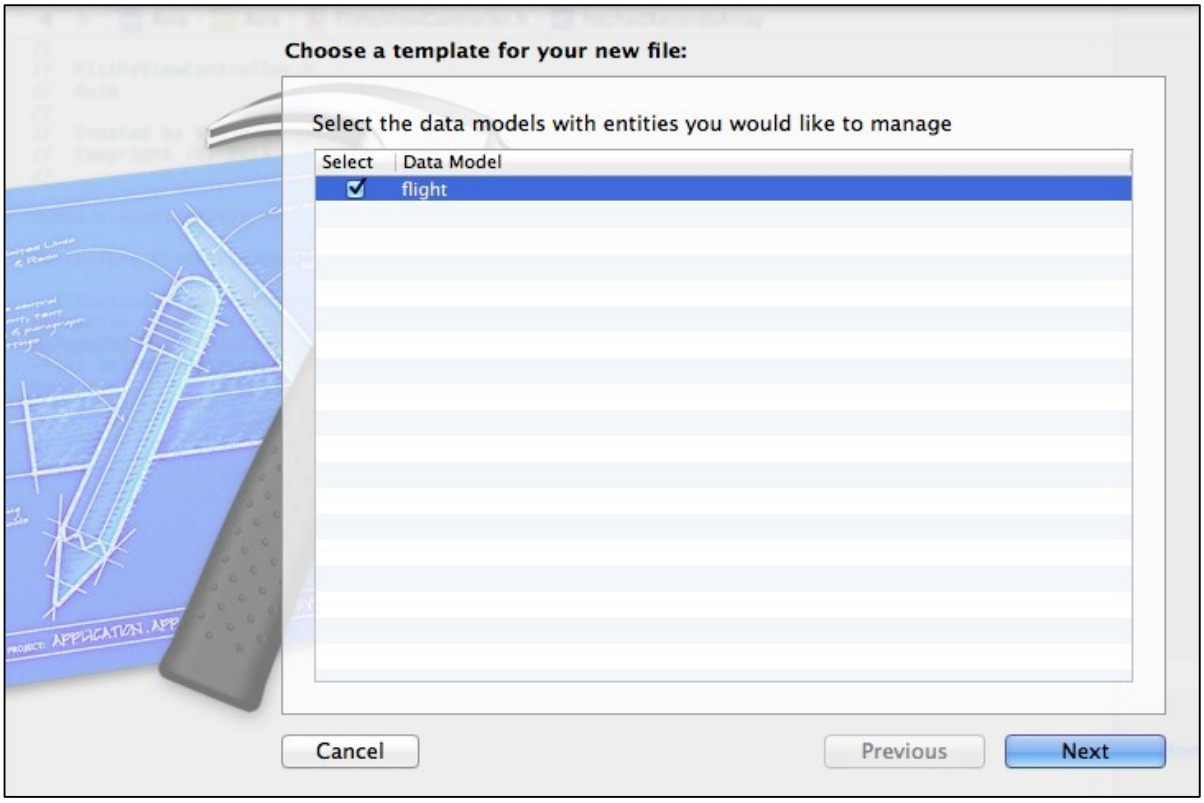


Рис. 13. Выбор шаблона

Выберите созданную нами сущность Record (Рис. 14).

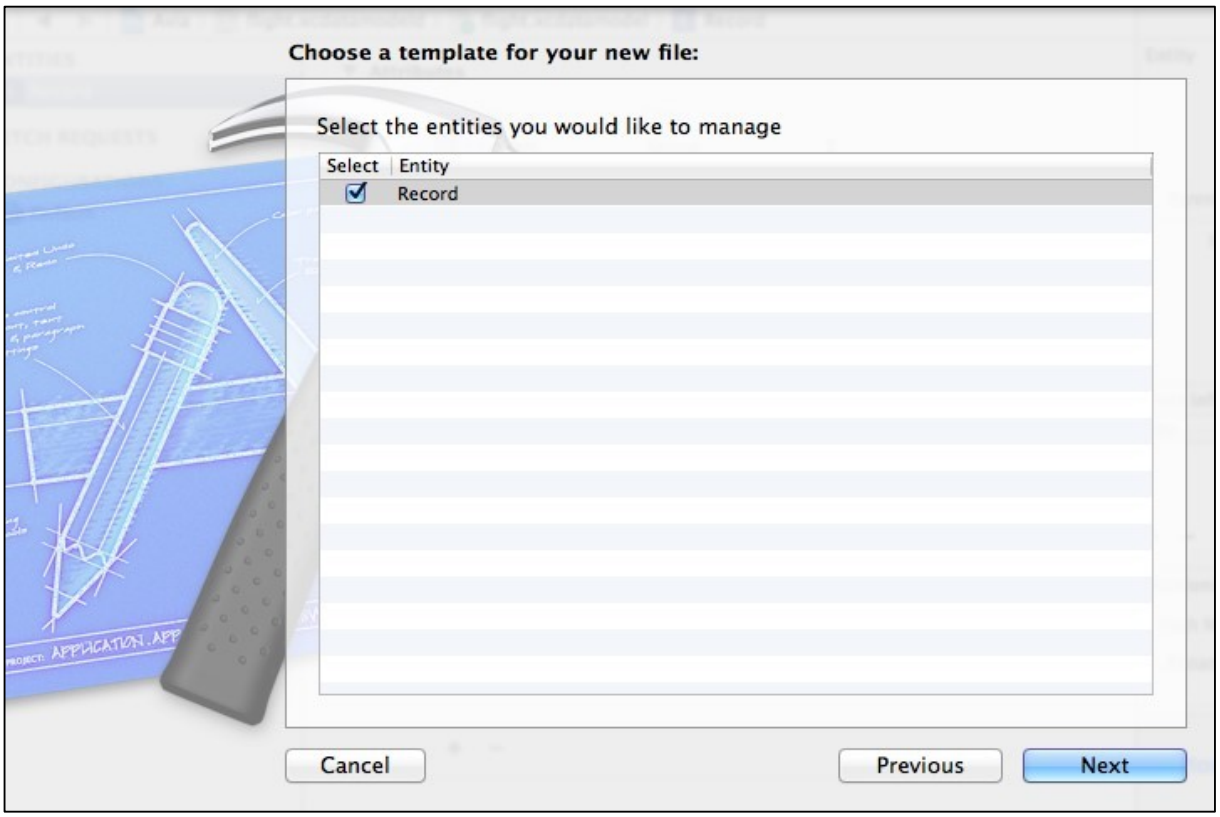


Рис. 14. Выбор сущности

Автоматически генерируется класс Record (Листинг 6).

```
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

@interface Record : NSManagedObject

@property (nonatomic, retain) NSString * cityfrom;
@property (nonatomic, retain) NSString * cityto;
@property (nonatomic, retain) NSString * company; @property
(nonatomic, retain) NSNumber * price;

@end
```

Листинг 6. Класс Record

В AppDelegate.m подключите класс Record (Листинг 7).

```
#import "Model/Record+CoreDataClass.h"
```

Листинг 7. Подключение класса Record.

Там же добавим методы для сохранения базы данных и получения всех рейсов в базе (Листинг 28).

```
- (void)saveContext {
    NSManagedObjectContext *context = self.persistentContainer.viewContext;
    NSError *error = nil;
    if ([context hasChanges] && ![context save:&error]) {
        // Replace this implementation with code to handle the error
appropriately.
        // abort() causes the application to generate a crash log and
terminate. You should not use this function in a shipping application,
although it may be useful during development.
        NSLog(@"Unresolved error %@, %@", error, error.userInfo);
        abort();
    }
}
```

Листинг 8. Сохранение базы данных

В методе класса AppDelegate.m необходимо добавить описание конструктора и деструктора (Листинг 9).

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:
(NSDictionary *)launchOptions {
  // Override point for customization after application launch.
```

Листинг 9. Конструктор и деструктор

И добавленном в листинге 9 метод описать заполнение базы данных при первом запуске приложения (Листинг 10).

```
if (![[NSUserDefaults standardUserDefaults] boolForKey:@"HasLaunchedOnce"])
{
        [[NSUserDefaults standardUserDefaults] setBool: YES
forKey:@"HasLaunchedOnce"];
        [[NSUserDefaults standardUserDefaults] synchronize];
        Record* firstFlight = [NSEntityDescription
insertNewObjectForEntityForName:@"Record"
inManagedObjectContext:self.managedObjectContext];

        firstFlight.cityFrom = @"Челябинск";
        firstFlight.cityTo = @"Москва";
        firstFlight.aviaCompany = @"Аэрофлот";
        firstFlight.price = 1000.0;

        Record* secondFlight = [NSEntityDescription
insertNewObjectForEntityForName:@"Record"
```

```
inManagedObjectContext:self.managedObjectContext];

        secondFlight.cityFrom = @"Челябинск";
        secondFlight.cityTo = @"Москва";
        secondFlight.aviaCompany = @"ЧелАвиа";
        secondFlight.price = 2000.0;

        Record* thirdFlight = [NSEntityDescription
insertNewObjectForEntityForName:@"Record"
inManagedObjectContext:self.managedObjectContext];

        thirdFlight.cityFrom = @"Екатеринбург";
        thirdFlight.cityTo = @"Уфа";
        thirdFlight.aviaCompany = @"Аэрофлот";
        thirdFlight.price = 500.0;

        Record* fourthFlight = [NSEntityDescription
insertNewObjectForEntityForName:@"Record"
inManagedObjectContext:self.managedObjectContext];

        fourthFlight.cityFrom = @"Челябинск";
        fourthFlight.cityTo = @"Уфа";
        fourthFlight.aviaCompany = @"РусЛайн";
        fourthFlight.price = 1500.0;

        Record* fifthFlight = [NSEntityDescription
insertNewObjectForEntityForName:@"Record"
inManagedObjectContext:self.managedObjectContext];

        fifthFlight.cityFrom = @"Екатеринбург";
        fifthFlight.cityTo = @"Москва";
        fifthFlight.aviaCompany = @"Аэрофлот";
        fifthFlight.price = 800.0;

        [self saveContext];
    }
```

Листинг 10. Заполнение базы данных

Возвращаемся к интерфейсу приложения. Выберите Main.storyboard

Добавьте в интерфейс приложения два компонента UITextField (Рис. 15).

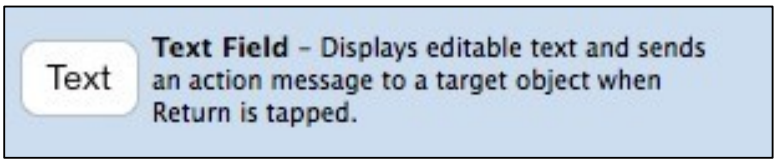


Рис. 15. Компонент UITextField

Создайте IBOutlet для каждого текстового поля в контроллере. Удерживая нажатой клавишу Ctrl, перетащите UITextField в область interface файла ViewController.h используя режим Assistant editor (см. инструкцию выше). Назовите поля cityFrom и cityTo. В интерфейсе класса должны появиться следующие строки (Листинг 11).

```
@property (weak, nonatomic) IBOutlet UITextField *cityFrom;
@property (weak, nonatomic) IBOutlet UITextField *cityTo;
```

Листинг 11. Описание пунктов отбытия и прибытия

В Interface Builder установите в свойство Placeholder для каждого текстового поля значения From и To (Рис. 16)

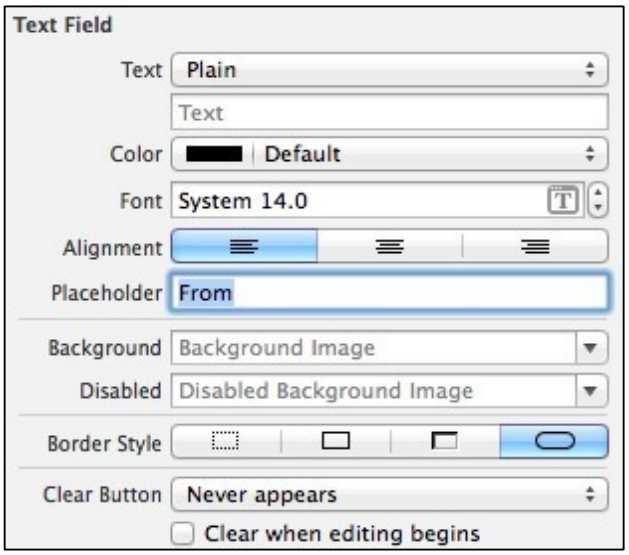


Рис. 16. Свойства текстового поля

Также следует добавить элемент Button (Рис. 17).

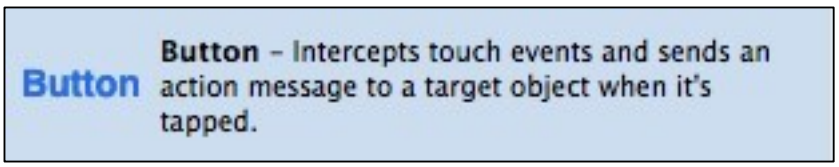


Рис. 17. Элемент Button

Назовите элемент Find (Рис. 18).

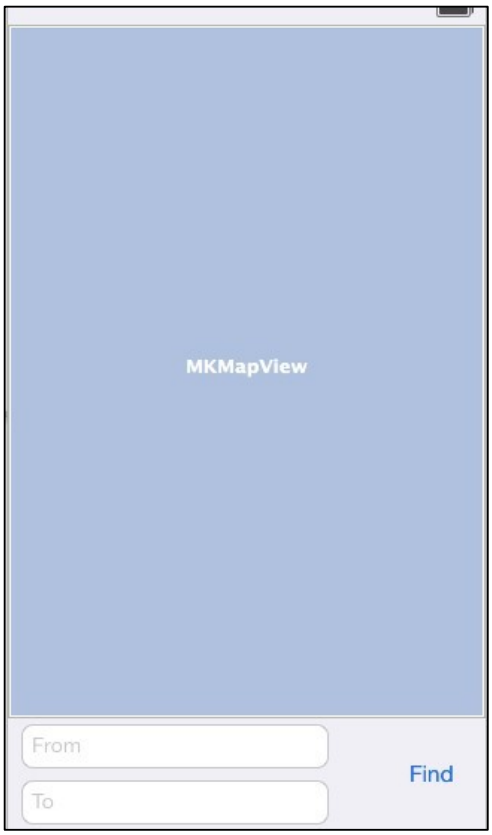


Рис. 18. Интерфейс программы

Создайте обработчик события нажатия кнопки (Touch Up Inside). Удерживая нажатой клавишу Ctrl перетащите событие Touch Up Inside в область interface файла ViewController.h используя режим Assistant editor.

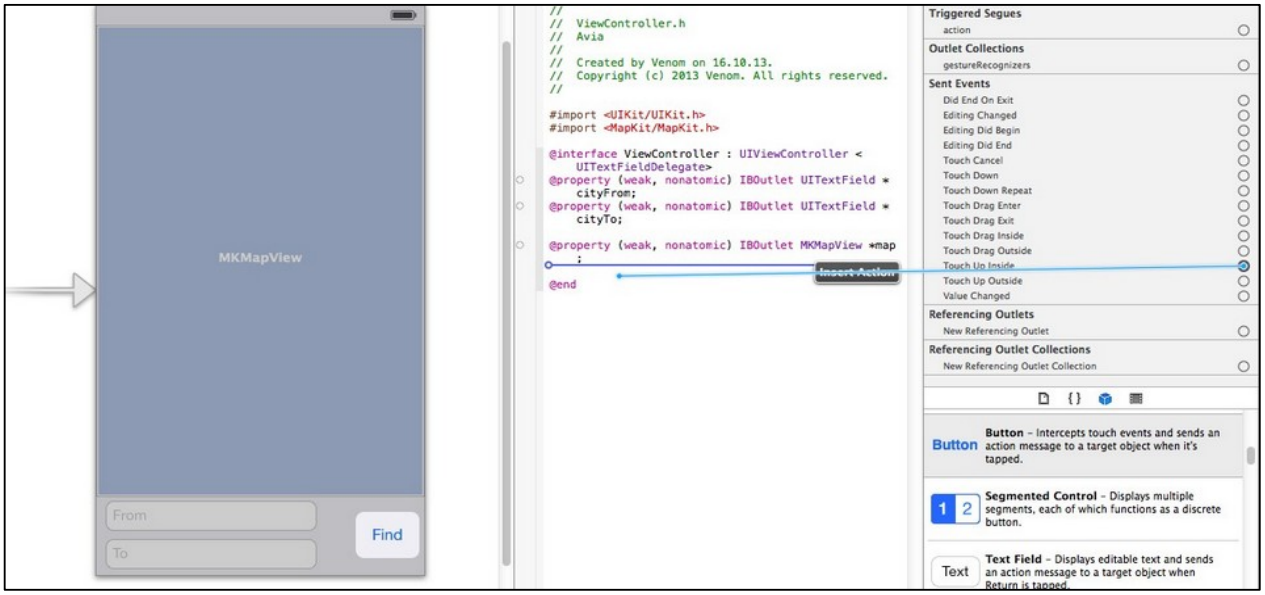


Рис. 19. Создание обработчика событий

Введите showFlights и нажмите Connect (Рис. 19).

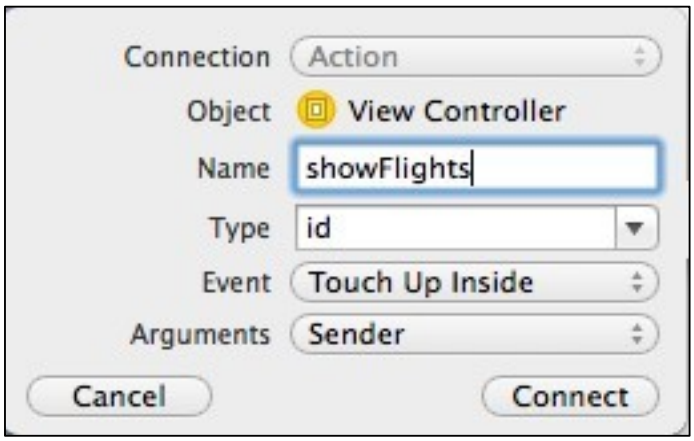


Рис. 20. Установка свойств обработчика

В область interface добавится описание обработчика (Листинг 12).

```
- (IBAction)showFlights:(id)sender
```

Листинг 12. Сигнатура обработчика

Переходим к классу ViewController в файле ViewController.h. Необходимо добавить делагат UITextFieldDelegate к нашему классу (Листинг 13).

```
@interface ViewController : UIViewController <UITextFieldDelegate>
```

Листинг 13**.** Делегат UITextFieldDelegate

Переходим к ViewController.m. Добавляем в интерфейс свойства пунктов отправки и прибытия (Листинг 14).

```
@interface ViewController ()
{
    int isCity;
    MKPointAnnotation *annotationFrom;
    MKPointAnnotation *annotationTo;
}
```

Листинг 14. Интерфейс ViewController

Переменные annotatiionFrom и annotatiionTo нужны для установки соответствующих отметок на карте. isCity нужна для определения какую именно отметку поставить.

В методе ViewDidLoad необходимо добавить обработчик длинного нажатия карты для установки отметки на карте (Листинг 15).

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
```

```
    UILongPressGestureRecognizer *longPressGesture =
[[UILongPressGestureRecognizer alloc] initWithTarget:self
action:@selector(handleLongPressGesture:)];
    [self.map addGestureRecognizer:longPressGesture];
}
```

Листинг 15**.** Обработчик длинного нажатия

При длинном нажатии на карте будет вызываться метод handleLongPressGesture (Листинг 16).

```
 -(void)handleLongPressGesture:(UIGestureRecognizer*)sender {
    if (sender.state == UIGestureRecognizerStateEnded) {
        CGPoint point = [sender locationInView:self.map];
        CLGeocoder *geocoder = [[CLGeocoder alloc] init];
        CLLocationCoordinate2D coord = [self.map convertPoint:point
toCoordinateFromView:self.map];
        CLLocation *location = [[CLLocation alloc]
initWithLatitude:coord.latitude longitude:coord.longitude];
        [geocoder reverseGeocodeLocation:location completionHandler:^(NSArray
*placemarks, NSError *error) {
            if (error) {
                NSLog(@"Geocode failed with error: %@", error);
                return;
            }

            for (CLPlacemark * placemark in placemarks) {
                [self setAnnotationToMap: self-
>isCity :placemark.locality:coord];
            }
        }];
    }
}
```

Листинг 16**.** Обработчик длинного нажатия

В этом методе определяются координаты (широта и долгота) нажатия и по ним определяется город нажатия.

Далее вызывается метод для установки метки (Листинг 17).

```
-(void)setAnnotationToMap:(int)type : (NSString *)title :
(CLLocationCoordinate2D)coordinate {
    if (type == 0) {
        [self.map removeAnnotation:annotationFrom];
        annotationFrom = [[MKPointAnnotation alloc] init];
        annotationFrom.title = title;
        annotationFrom.coordinate = coordinate;
        [self.map addAnnotation:annotationFrom];
        self.cityFrom.text = title;
    }
}
```

Листинг 17**.** Установка отметки

По умолчанию отметка будет ставиться в город отправления. Чтобы поменять отметку на город прибытия нужно нажать на соответствующее текстовое поле (Листинг 18).

```
-(void)textFieldDidBeginEditing:(UITextField * )textField {
    if (textField == self.cityFrom)
        isCity = 0;
    else if (textField == self.cityTo)
        isCity = 1;

    [textField resignFirstResponder];
}
```

Листинг 18. Установка пункта прибытия

Опишем вызов контоллера при нажатии ранее созданной кнопки (Листинг 19).

```
- (IBAction)showFlights:(id)sender
{
    FlithsViewController *flights =[[FlithsViewController
alloc]initWithBothCity:self.textFrom.text :self.textTo.text];
    [self presentViewController:flights animated:YES completion:nil];
}
```

Листинг 19. Вызов контроллера

Добавить обработку маршрутов самостоятельно!

1.6. IOS ПРИЛОЖЕНИЕ 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.

Указания к работе:

Изучить пример создания приложения, которое позволит хранить список студентов 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.

Перед выполнением просмотрите видео https://www.youtube.com/watch?v=2TYvcdBPMB4 и в ходе работы внесите изменения в приведенный код в соответствии с синтаксисом языка Swift3.

Ход работы:

1.6.1. Создание проекта Xcode.

Создаем проект iOS – Application – Single View Application, но, на этот раз, ставим галочку напротив Use Core Data (рис.22).

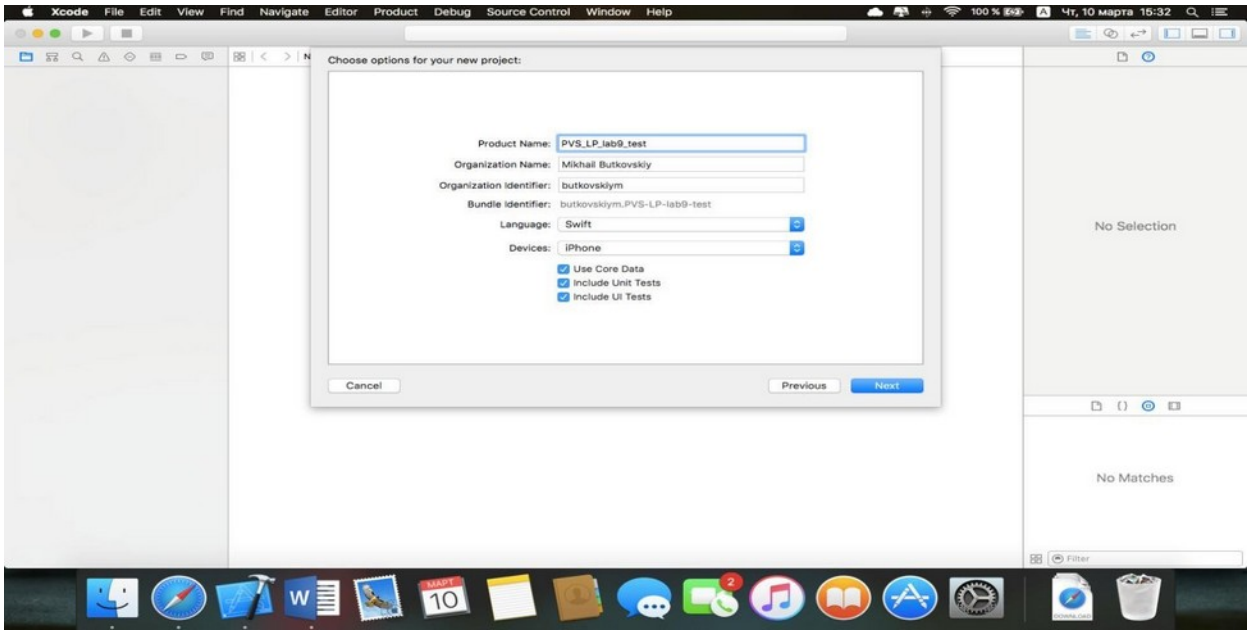


Рис.22. Создание проекта

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.23).

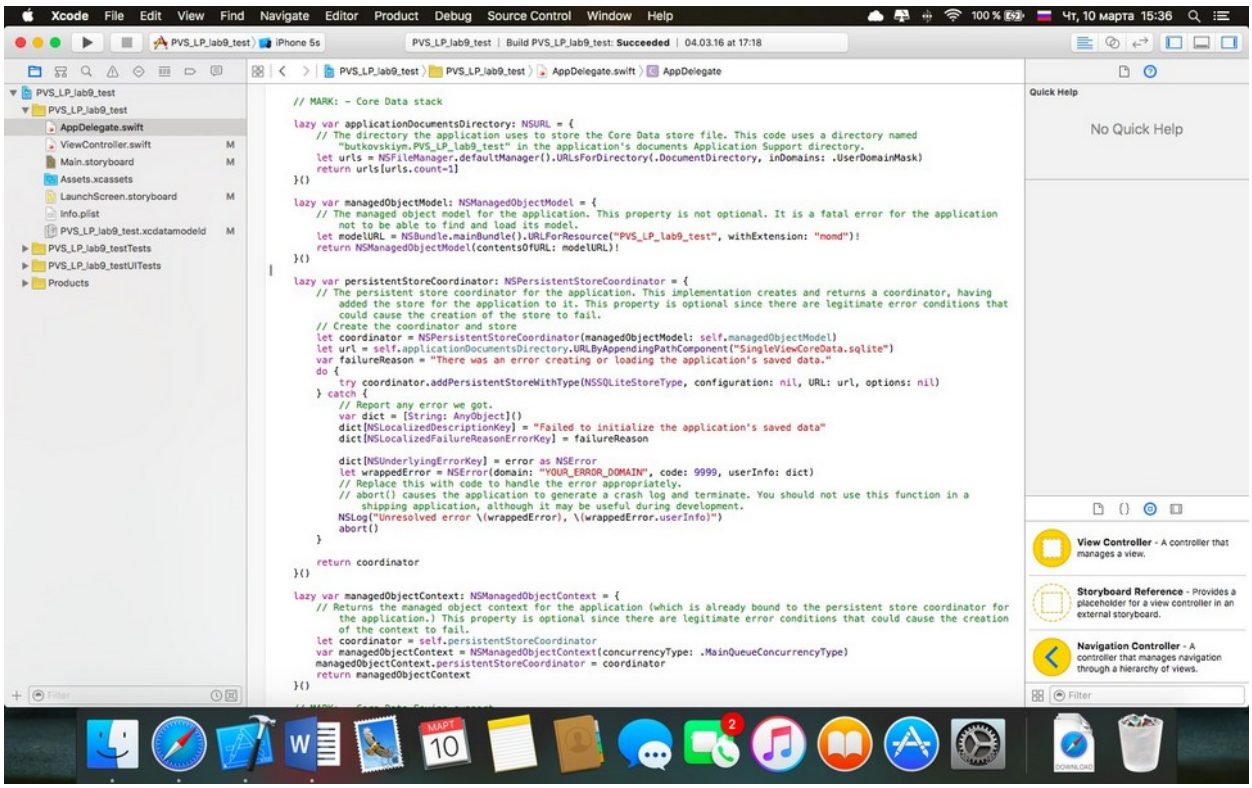


Рис.23. Описание класса AppDelegate

1.6.2 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.24). Ячейке Table View Cell присваиваем идентификатор: “studentCell”. Текстовому полю задаем Placeholder: “Введите Фамилию и Имя студента”.

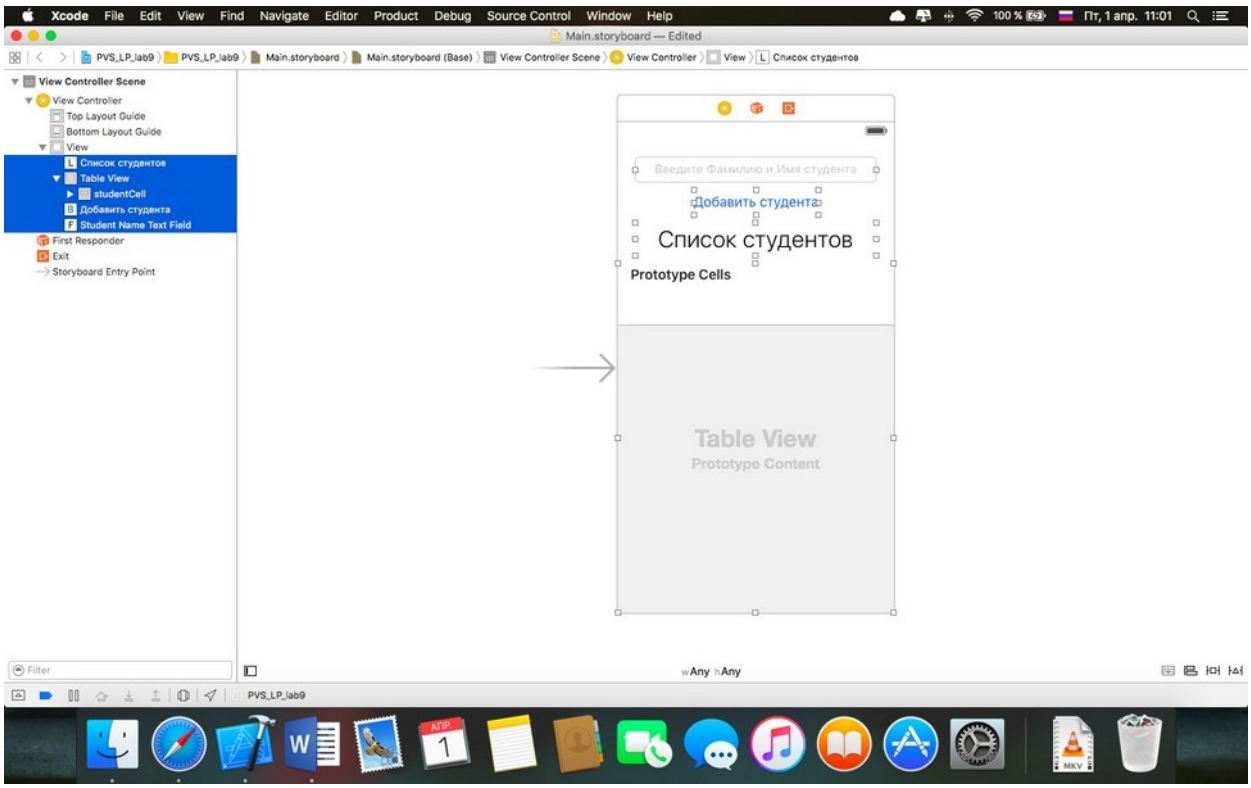


Рис. 24. Добавление объекта Label

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.25).

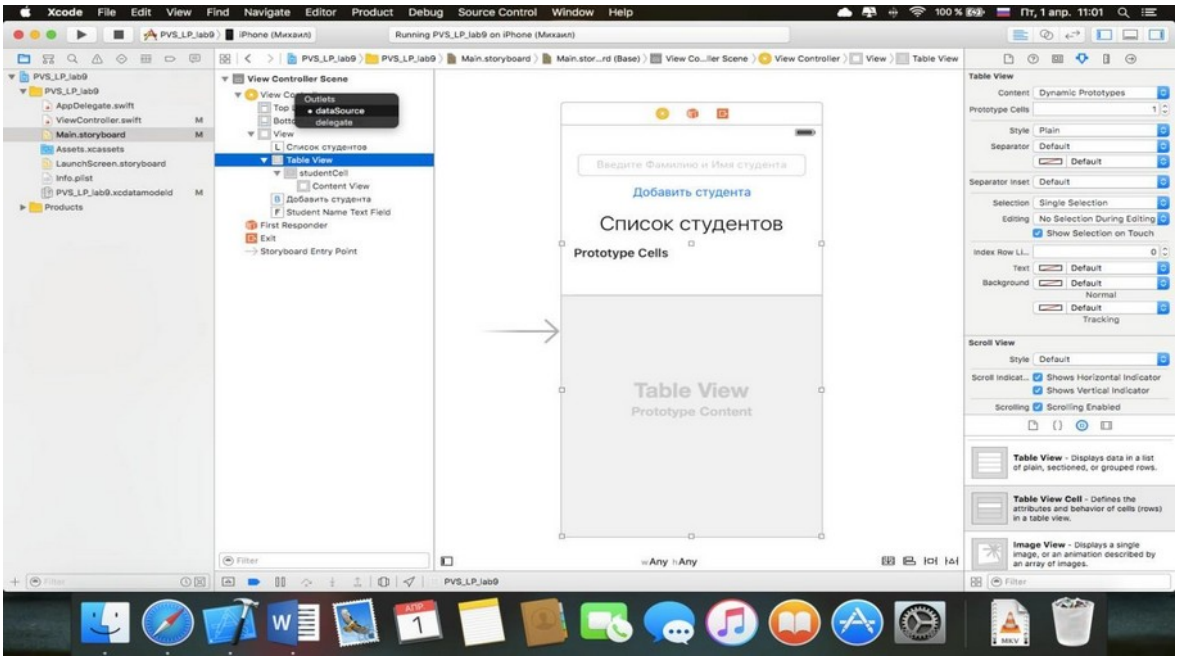


Рис. 25. Дерево объектов интерфейса

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){ }

1.6.3 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. 26).

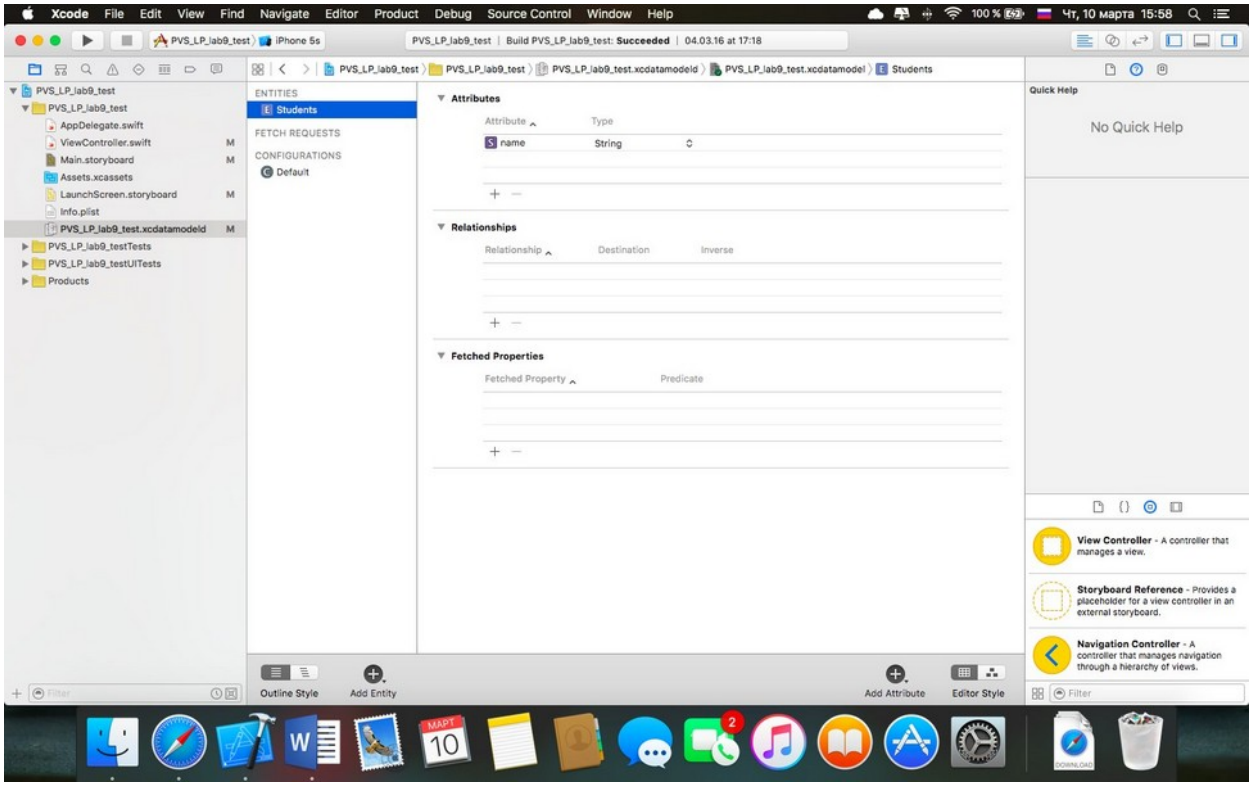


Рис. 26. Файл с расширением .xcdatamodeld

1.6.4 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: UITableViewCellEditingStyle, 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 }
```

Рис.27. Проверка работы приложения

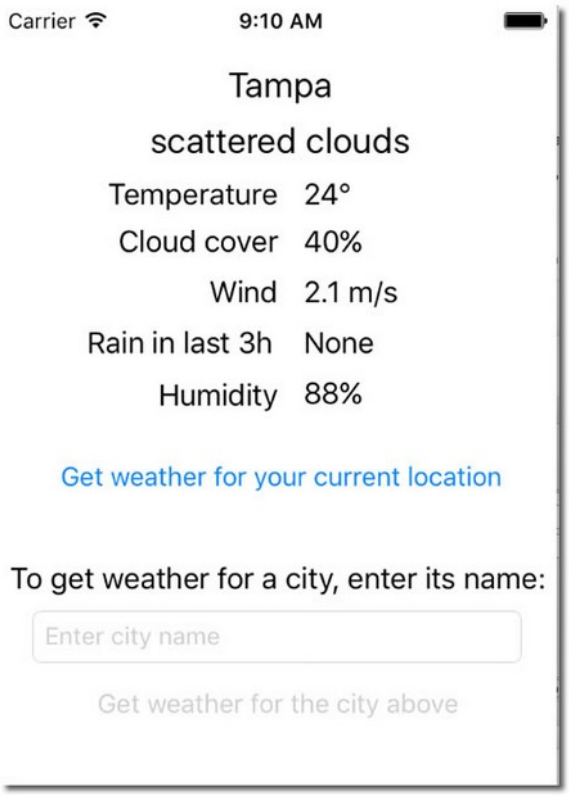
1.7. iOS ПРИЛОЖЕНИЕ ПРОГНОЗА ПОГОДЫ

Цель: изучить пример разработки приложения *Прогноз погоды*

Разобрать примеры из статей:

- Статья 1 — описание сервиса погоды OpenWeatherMap, его API, и как получить данные погоды для конкретного города вручную или программно.
- Статья 2 — использование сетевых возможностей и класса `NSURLSession`. Преобразование данных о погоде, полученных от сервиса OpenWeatherMap в виде строк в формате JSON, в элементы словаря (dictionary)

- Статья 3 — разработка интерфейса приложения.
- Статья 4 — пример приложения с геолокацией (из примера 1.3).
- Статья 5 — реализация функционала: 1) автоматическое определение местоположения пользователя и вывод прогноза погоды для текущего положения пользователя, 2) Получение данных о прогнозе погоды для текущего местоположения при нажатии кнопки **Get weather for your current location** , 3) Получение прогноза погоды по названию города, введеного в строке **Get weather for the city above**, и нажатию кнопки.



2. ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

2.1. Создание iOS приложения с хранение данных в NSUserDefaults и файле .plist

Создать iOS приложение на языке Swift с интерфейсом авторизации и храненением данных в NSUserDefaults, а так же выводом данных, хранимых в файле .plist, используя представление UICollectionView для вывода данных, согласно варианту задания.

Приложение должно поддерживать локализацию на 3 языка, два из которых русский и английский. Третьим языком можем быть белорусский, польский, французский, немецкий, итальянский и др.

Код проекта распределить по группам (каталогам) согласно концепции MVC (Model, View, Controller).

2.2. Создание iOS приложения с использованием MapKit, CoreLocation, CoreData

Создать iOS приложение на языке Swift согласно варианту задания. Для макета списка использовать UITableView. При создании приложения использовать библиотеки MapKit, CoreLocation, CoreData и сервис прогноза погоды согласно варианту задания.

Приложение должно поддерживать локализацию на 3 языка, два из которых русский и английский. Третьим языком можем быть белорусский, польский, французский, немецкий, итальянский и др.

Код проекта распределить по группам (каталогам) согласно концепции MVC (Model, View, Controller) или MVVM (Model, View, ViewModel).

ВАРИАНТЫ ЗАДАНИЙ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

№	***Вариант к заданию 2.1***	***Вариант к заданию 2.2.***
1	Приложение «Библиотека» выводит список книг и изображений книг. При выборе книги пользователь может просмотреть ее детальное описание.	Требуется разработать программу бронирования билетов на маршрутный автобус Гродненской области, используя CoreData, CoreLocation и MapKit. Пользователь выбирает маршрут на карте, и ему предлагается список городов, при выборе города — список рейсов, включающих номер маршрутного автобуса, стоимость проезда и наличие свободных мест. При выборе города на карте пользователь может посмотреть прогноз погоды, получая данные от сервиса https://openweathermap.org/api. Данные должны храниться в базе данных sqlite, доступ через CoreData.
2	Приложение «Фильмотека» выводит список фильмов и заставок-изображений фильмов. При выборе фильма пользователь может просмотреть его детальное описание.	Требуется разработать программу интерактивную карту музеев города Гродно, используя CoreData, CoreLocation и MapKit. Пользователь выбирает район города на карте, и ему предлагается список музеев в данном районе и выставки, которые в каждом музее проходят. При выборе района города на карте пользователь может посмотреть прогноз погоды, получая данные от сервиса https://openweathermap.org/api. Данные должны храниться в базе данных sqlite, доступ через CoreData.
3	Приложение «Аудиотека» выводит список альбомов и заставок-изображений	Требуется разработать программу интерактивную карту парков города Минска, используя CoreData, CoreLocation и MapKit. Пользователь выбирает район города на карте, и ему предлагается список

№	*Вариант к заданию 2.1*	*Вариант к заданию 2.2.*
	альбомов. При выборе альбома пользователь может просмотреть его детальное описание.	парков в данном районе и мероприятия, которые в каждом парке проходят. При выборе района города на карте пользователь может посмотреть прогноз погоды, получая данные от сервиса https://tech.yandex.ru/weather/. Данные должны храниться в базе данных sqlite, доступ через CoreData.
4	Приложение «Контакты» выводит список контактов и их фото. При выборе контакта пользователь может просмотреть его детальное описание (ФИО, адрес, номер телефона, адрес электронной почты и др.).	Требуется разработать программу бронирования билетов на самолет, используя CoreData, CoreLocation и MapKit. Пользователь выбирает маршрут на карте, и ему предлагается список аэропортов Беларуси, при выборе аэропорта — список рейсов, включающих название рейса, стоимость проезда и наличие свободных мест. При выборе города на карте пользователь может посмотреть прогноз погоды, получая данные от сервиса https://tech.yandex.ru/weather/. Данные должны храниться в базе данных sqlite, доступ через CoreData.
5	Приложение «Онлайн-курсы» выводит список онлайн-курсов и изображение заставку для каждого курса. При выборе курса пользователь может просмотреть его детальное описание, включая программу из 10 пунктов, сведения о преподавателе.	Требуется разработать программу бронирования билетов на автобус, используя CoreData, CoreLocation и MapKit. Пользователь выбирает маршрут на карте, и ему предлагается список рейсов, включающих название автостанции, стоимость проезда и наличие свободных мест. Если в городе несколько автостанций, показывать данные о всех. При выборе города на карте пользователь может посмотреть прогноз погоды, получая данные от сервиса openweathermap.org/api. Данные должны храниться в базе данных sqlite, доступ через CoreData.
6	Приложение «Аптека» выводит список лекарств и изображение каждого лекарства. При выборе курса пользователь может просмотреть его детальное описание, включая состав.	Требуется разработать программу бронирования авиабилетов из Минска в Киев, Вильнюс и Смоленск, используя CoreData, CoreLocation и MapKit. Пользователь выбирает маршрут на карте, и ему предлагается список рейсов, включающих название авиакомпании, стоимость перелёта. При выборе аэропорта на карте пользователь может посмотреть прогноз погоды, получая данные от сервиса https://openweathermap.org/api. Данные должны храниться в базе данных sqlite, доступ

№	*Вариант к заданию 2.1*	*Вариант к заданию 2.2.*
		через CoreData.
7	Приложение «Зоомагазин» выводит список кормов для животных и изображение. При выборе корма в списке пользователь может просмотреть его детальное описание, включая состав.	Требуется разработать программу интерактивную карту факультетов и общежитий БГУИР, используя CoreData, CoreLocation и MapKit. Пользователь выбирает район города на карте, и ему предлагается список факультетов и общежитий БГУИР. При выборе района города на карте пользователь может посмотреть прогноз погоды, получая данные от сервиса https://www.wunderground.com/weather/api/?MR=1 . Данные должны храниться в базе данных sqlite, доступ через CoreData.
8	Приложение «БГУ» выводит список факультетов и логотип факультета. При выборе факультета пользователь может просмотреть его детальное описание, включая информацию о декане, дате основания и перечне специальностей.	Требуется разработать программу интерактивную карту вузов и колледжей, лицеев города Бреста, используя CoreData, CoreLocation и MapKit. Пользователь выбирает район города на карте, и ему предлагается список учебных заведений в данном районе города. При выборе района города на карте пользователь может посмотреть прогноз погоды, получая данные от сервиса https://tech.yandex.ru/weather/. Данные должны храниться в базе данных sqlite, доступ через CoreData.
9	Приложение «ФПМИ» выводит список кафедр и изображение-заставку кафедры. При выборе кафедры пользователь может просмотреть ее детальное описание, включая заведующего, список читаемых курсов.	Требуется разработать программу интерактивную карту вузов и колледжей, лицеев города Гродно, используя CoreData, CoreLocation и MapKit. Пользователь выбирает район города на карте, и ему предлагается список учебных заведений в данном районе города. При выборе района города на карте пользователь может посмотреть прогноз погоды, получая данные от сервиса https://openweathermap.org/api. Данные должны храниться в базе данных sqlite, доступ через CoreData.
10	Приложение «Учебники» выводит список учебников, например для 10 класса и изображение каждого учебника. При выборе учебника пользователь может	Требуется разработать программу интерактивную карту музеев Могилевской области (за исключением Могилева), используя CoreData, CoreLocation и MapKit. Пользователь выбирает город на карте, и ему предлагается список музеев в данном городе и выставки, которые в каждом музее проходят. При выборе города на карте

№	*Вариант к заданию 2.1*	*Вариант к заданию 2.2.*
	просмотреть его детальное описание, включая авторов и список тем, изложенных в учебнике.	пользователь может посмотреть прогноз погоды получая данные от сервиса https://tech.yandex.ru/weather/. Данные должны храниться в базе данных sqlite, доступ через CoreData.
11	Приложение «Белорусские поэты» выводит список белорусских поэтов, фотографию каждого. При выборе поэта пользователь может просмотреть его биографию и список произведений.	Требуется разработать программу интерактивную карту музеев Витебской области, используя CoreData, CoreLocation и MapKit. Пользователь выбирает город на карте, и ему предлагается список музеев в данном городе и выставки, которые в каждом музее проходят. При выборе города на карте пользователь может посмотреть прогноз погоды получая данные от сервиса https://developer.yahoo.com/weather/?guccounter=1. Данные должны храниться в базе данных sqlite, доступ через CoreData.
12	Приложение «Белорусские писатели» выводит список белорусских писателей, фотографию каждого. При выборе писателя пользователь может просмотреть его биографию и список произведений.	Требуется разработать программу интерактивную карту Гомельской области, используя CoreData, CoreLocation и MapKit. Пользователь выбирает город на карте, и ему предлагается список историческая справа о городе, а так же как доехать в выбранный город из Минска. При выборе города на карте пользователь может посмотреть прогноз погоды, получая данные от сервиса https://tech.yandex.ru/weather/. Данные должны храниться в базе данных sqlite, доступ через CoreData.
13	Приложение «Кафедра ТП» выводит список преподавателей кафедры и их фото (использовать фото с сайта факультета). При выборе преподавателя пользователь может просмотреть информацию о нем и список читаемых дисциплин (курсов).	Требуется разработать программу интерактивную карту музеев города Минска, используя CoreData, CoreLocation и MapKit. Пользователь выбирает район города на карте, и ему предлагается список музеев в данном районе и выставки, которые в каждом музее проходят. При выборе района города на карте пользователь может посмотреть прогноз погоды, получая данные от сервиса https://www.wunderground.com/weather/api/?MR=1. Данные должны храниться в базе данных sqlite, доступ через CoreData.
14	Приложение «Белорусская кухня»	Требуется разработать программу бронирования билетов на железнодорожный транспорт,

№	***Вариант к заданию 2.1***	***Вариант к заданию 2.2.***
	выводит список белорусских блюд, фотографию каждого. При выборе блюда пользователь может просмотреть рекомендации по его приготовлению и список необходимых продуктов.	используя CoreData, CoreLocation и MapKit. Пользователь выбирает маршрут на карте, и ему предлагается список железнодорожных вокзалов, при выборе вокзала — список рейсов, включающих название поезда, стоимость проезда и наличие свободных мест. При выборе города на карте пользователь может посмотреть прогноз погоды, получая данные от сервиса https://developer.yahoo.com/weather/?guccounter=1. Данные должны храниться в базе данных sqlite, доступ через CoreData.
15	Приложение «Музеи Минска» выводит список музеев Минска, фотографию каждого. При выборе музея пользователь может просмотреть сведения о нем, время работы, контакты и сведения о директоре музея.	Требуется разработать программу бронирования билетов на автобус, используя CoreData, CoreLocation и MapKit. Пользователь выбирает маршрут на карте, и ему предлагается список рейсов, включающих название автостанции, стоимость проезда и наличие свободных мест. Если в городе несколько автостанций, показывать данные о всех. При выборе города на карте пользователь может посмотреть прогноз погоды, получая данные от сервиса https://tech.yandex.ru/weather/. Данные должны храниться в базе данных sqlite, доступ через CoreData.
16	Приложение «Памятники Минска» выводит список памятников Минска, фотографию каждого. При выборе памятника пользователь может просмотреть сведения о нем, данные о местонахождении и сведения о скульпторе.	Требуется разработать программу бронирования авиабилетов, используя CoreData, CoreLocation и MapKit. Пользователь выбирает маршрут на карте, и ему предлагается список рейсов, включающих название авиакомпании, стоимость перелёта. При выборе аэропорта на карте пользователь может посмотреть прогноз погоды, получая данные от севрса https://openweathermap.org/api. Данные должны храниться в базе данных sqlite, доступ через CoreData.
17	Приложение «Минская область» выводит список городов Минской области, фотографию каждого или герб города. При выборе города пользователь может просмотреть сведения о нем, данные о	Требуется разработать программу интерактивную карту факультетов и общежитий БГУ, используя CoreData, CoreLocation и MapKit. Пользователь выбирает район города на карте, и ему предлагается список факультетов и общежитий БГУ. При выборе района города на карте пользователь может посмотреть прогноз погодыполучая данные от сервиса

№	*Вариант к заданию 2.1*	*Вариант к заданию 2.2.*
	местонахождении и известные исторические факты.	https://www.wunderground.com/weather/api/?MR=1 . Данные должны храниться в базе данных sqlite, доступ через CoreData.
18	Приложение «Музеи Витебска» выводит список музеев Витебска и Витебской области, фотографию каждого. При выборе музея пользователь может просмотреть сведения о нем, время работы, контакты и сведения о директоре музея.	Требуется разработать программу интерактивную карту вузов и колледжей, лицеев города Бреста, используя CoreData, CoreLocation и MapKit. Пользователь выбирает район города на карте, и ему предлагается список учебных заведений в данном районе города. При выборе района города на карте пользователь может посмотреть прогноз погоды, получая данные от сервиса https://tech.yandex.ru/weather/. Данные должны храниться в базе данных sqlite, доступ через CoreData.
19	Приложение «Памятники Витебска» выводит список памятников Витебска, фотографию каждого. При выборе памятника пользователь может просмотреть сведения о нем, данные о местонахождении и сведения о скульпторе.	Требуется разработать программу интерактивную карту музеев Минской области (за исключением Минска), используя CoreData, CoreLocation и MapKit. Пользователь выбирает город на карте, и ему предлагается список музеев в данном городе и выставки, которые в каждом музее проходят. При выборе города на карте пользователь может посмотреть прогноз погоды получая данные от севрса https://openweathermap.org/api. Данные должны храниться в базе данных sqlite, доступ через CoreData.
20	Приложение «Озера Беларуси» выводит список озер, фотографию каждого. При выборе озера пользователь может просмотреть сведения о нем, данные о местонахождении и известные исторические факты.	Требуется разработать программу интерактивную карту музеев Витебской области, используя CoreData, CoreLocation и MapKit. Пользователь выбирает город на карте, и ему предлагается список музеев в данном городе и выставки, которые в каждом музее проходят. При выборе города на карте пользователь может посмотреть прогноз погоды получая данные от сервиса https://developer.yahoo.com/weather/?guccounter=1. Данные должны храниться в базе данных sqlite, доступ через CoreData.
21	Приложение «Санатории Беларуси» выводит список санаториев, фотографию каждого. При выборе санатория	Требуется разработать программу интерактивную карту Гомельской области, используя CoreData, CoreLocation и MapKit. Пользователь выбирает город на карте, и ему предлагается список историческая справа о городе, а так же как доехать

№	***Вариант к заданию 2.1***	***Вариант к заданию 2.2.***
	пользователь может просмотреть сведения о нем, данные о местонахождении и основные направления деятельности.	в выбранный город из Минска. При выборе города на карте пользователь может посмотреть прогноз погоды, получая данные от сервиса https://tech.yandex.ru/weather/. Данные должны храниться в базе данных sqlite, доступ через CoreData.
22	Приложение «Парки Минска» выводит список парков Минска, фотографию каждого. При выборе парка пользователь может просмотреть сведения о нем, время работы, контакты и интересные исторические факты	Требуется разработать программу интерактивную карту Могилевской области, используя CoreData, CoreLocation и MapKit. Пользователь выбирает город на карте, и ему предлагается список историческая справа о городе, а так же как доехать в выбранный город из Минска. При выборе города на карте пользователь может посмотреть прогноз погоды получая данные от сервиса https://openweathermap.org/api. Данные должны храниться в базе данных sqlite, доступ через CoreData.
23	Приложение «Музеи Гомеля» выводит список музеев Гомеля и Гомельской области, фотографию каждого. При выборе музея пользователь может просмотреть сведения о нем, время работы, контакты и сведения о директоре музея.	Требуется разработать программу бронирования билетов на междугородних автобусов Брестской области, используя CoreData, CoreLocation и MapKit. Пользователь выбирает маршрут на карте, и ему предлагается список автовокзалов, при выборе вокзала — список рейсов, включающих название автобуса, стоимость проезда и наличие свободных мест. При выборе города на карте пользователь может посмотреть прогноз погоды, получая данные от сервиса https://developer.yahoo.com/weather/?guccounter=1. Данные должны храниться в базе данных sqlite, доступ через CoreData.

№	***Вариант к заданию 2.1***	***Вариант к заданию 2.2.***
24	Приложение «Дудутки», которое является путеводителем по музейному комплексу и выводит список основных объектов комплекса, фотографию каждого. При выборе объекта пользователь может просмотреть сведения о нем, исторический факты и другие данные.	Требуется разработать программу бронирования билетов на маршрутный автобус Минской области, используя CoreData, CoreLocation и MapKit. Пользователь выбирает маршрут на карте, и ему предлагается список городов, при выборе города — список рейсов, включающих номер маршрутного автобуса, стоимость проезда и наличие свободных мест. При выборе города на карте пользователь может посмотреть прогноз погоды, получая данные от сервиса https://openweathermap.org/api. Данные должны храниться в базе данных sqlite, доступ через CoreData.
25	Приложение «Брестская крепость», которое является путеводителем по музейному комплексу и выводит список основных объектов комплекса, фотографию каждого. При выборе объекта пользователь может просмотреть сведения о нем, исторический факты и другие данные.	Требуется разработать программу бронирования билетов на на фестиваль «Славянский базар», используя CoreData, CoreLocation и MapKit. Пользователь выбирает точку проведения события (район) на карте города, и ему предлагается список ближайших площадок, при выборе площадки — список мероприятий, включающих дату и время проведения, стоимость билета и наличие свободных мест. При выборе района Витебска на карте и даты пользователь может посмотреть прогноз погоды, получая данные от сервиса https://openweathermap.org/api. Данные должны храниться в базе данных sqlite, доступ через CoreData.
26	Приложение «Славянский базар», которое является путеводителем по фестивалю и выводит список основных площадок фестиваля, фотографию каждой. При выборе объекта пользователь может просмотреть сведения о нем, исторические факты и другие данные.	Требуется разработать программу интерактивную карту музеев Пинска и ближайших к городу достопримечательностей, используя CoreData, CoreLocation и MapKit. Пользователь выбирает город/поселок/деревню на карте, и ему предлагается список музеев в данном населенном пункте и интересные факты о данном мезее. При выборе города на карте пользователь может посмотреть прогноз погоды получая данные от сервиса https://developer.yahoo.com/weather/?guccounter=1. Данные должны храниться в базе данных sqlite, доступ через CoreData.
27	Приложение «Музеи Могилева» выводит список музеев Могилева	Требуется разработать программу интерактивную карту вузов и колледжей, лицеев города Гродно, используя CoreData, CoreLocation и MapKit.

№	*Вариант к заданию 2.1*	*Вариант к заданию 2.2.*
	и Могилевской области, фотографию каждого. При выборе музея пользователь может просмотреть сведения о нем, время работы, контакты и сведения о директоре музея.	Пользователь выбирает район города на карте, и ему предлагается список учебных заведений в данном районе города. При выборе района города на карте пользователь может посмотреть прогноз погоды, получая данные от сервиса https://tech.yandex.ru/weather/. Данные должны храниться в базе данных sqlite, доступ через CoreData.
28	Приложение «Памятники Гродно» выводит список памятников Гродно, фотографию каждого. При выборе памятника пользователь может просмотреть сведения о нем, данные о местонахождении и сведения о скульпторе.	Требуется разработать программу интерактивную карту музеев белорусского Полесья, используя CoreData, CoreLocation и MapKit. Пользователь выбирает населенный пункт на карте, и ему предлагается список музеев в данном населенном пункте и выставки, которые в каждом музее проходят. При выборе населенного пункта на карте пользователь может посмотреть прогноз погоды получая данные от севрса https://openweathermap.org/api. Данные должны храниться в базе данных sqlite, доступ через CoreData.

КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ

1. Что такое CoreLocation?
2. Что такое MapKit?
3. Как сохранить или прочитать данные из NSUserDefaults?
4. Опишите файлы формата .plist и для каких задач используются?
5. Как в Swift осуществляется сетевое взаимодействие?
6. Из каких компонент состоит графический интерфейс программ для платформы iOS на Swift?
7. Какие настройки необходимо включить в файле настроек проекта для определения местоположения?
8. Что такое CoreData?
9. Как произвести вставку записей в базу данных?
10. Как произвести поиск по базе данных?

Библиографический список

1. Маскри. Молли. Топли, Ким. Марк. Дэвид. и др. Swift 3: разработка приложений в среде Xcode для iPhone и iPad с использованием IOS SDK. 3-е изд.: Пер. с англ. - СпБ.: ООО "Альфа-книга", 2017. - 896 с. : ил. - Парал. тит. англ.
2. Михаель Приват и Роберт Варнер «Pro Core Data for iOS» — http://davidhstannard.com/PDXJS/Pro_Core_Data_for_iOS.pdf
3. Core Data Programming Guide — https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreData/index.html