Задание 5: Top Places

Цель:

В этой домашней работе вы должны создать приложение, которое показывает пользователю список наиболее популярных мест на Земле, в которых сделаны фотографии и размещены на сервисе **Flickr**, а также дает возможность просмотра этих фотографий.
Первый пункт этого Задания состоит в создании таблично пользовательского интерфейса с двумя закладками: **Top Places** и **Recents**. Первая закладка показывает пользователю популярные места **Flickr** фотографий и дает возможность их просмотра. Вторая закладка дает пользователю доступ к “недавно просмотренным” внутри этого приложения фотографиям.
Цель данного задания- познакомиться с **Table Views**, **Scroll Views**, **Image View** и многопоточностью и узнать, как строить универсальные приложения, которые работают как на iPhone, так и на iPad (с соответствующим UI на каждом).
Все данные, которые вам необходимы, будут загружаться с Flickr.com с использованием API Flickr. Вам будет предоставлен код, который создает URLs для запросов к Flickr, которые вам понадобятся в этой домашней работе.
Обязательно посмотрите ниже раздел подсказок (**Подсказки (Hints)**)!

Материалы

- Это совершенно новое приложение, вам не понадобится ничего (кроме полученных знаний) из первых четырех домашних работ.

- Вам необходимо получить ключ Flickr API. Бесплатный Flickr account вполне допустим (так как вы не будете размещать на Flickr фотографии, а только запрашивать их).

- Вспомогательный класс FlickrFetcher очень полезен для этого задания.

Обязательные пункты задания

1. Загрузите данные с URL, поставляемого методом **URLForTopPlaces** вспомогательного класса **FlickrFetcher**, для получения массива наиболее популярных мест, где были сделаны **Flickr** фотографии за последнее время. Смотри (**Подсказки (Hints)**) как интерпретировать данные, возвращаемые **Flickr**.

2. Создайте **UITabBarController** пользовательский интерфейс с двумя закладками. Первая закладка показывает **UITableView** список мест, полученных в обязательном пункте 1, разделенных на секции по странам, а внутри секции по алфавиту. Вторая закладка показывает **UITableView** список из 20 недавно просмотренных (в вашем приложении) фотографий (в хронологическом порядке с поздним в качестве первого фото и без дупликатов).

3. При появлении популярного места в **UITableView** в вашем приложении, наиболее детальная часть местоположения (например, имя города) должна быть **title** ячейки (cell) **UITableView**, а остальная часть местоположения (например, штат, провинция и т.д.) должна появляться как **subtitle** ячейки **UITableView**. Название страны должно быть в заголовке секции.

4. Если пользователь выберет место в **UITableView**, вы должны опять запросить у **Flickr** 50 фотографий с этого места и показать их в списке. **URL** поставляется вспомогательным классом **FlickrFetcher** и методом класса **URLForPhotosInPlace: maxResult:** для получения этих фотографий из **Flickr**.

5. Любой список фотографий должен показывать заголовок фотографии как **title** ячейки **UITableView**, а описание (description) фотографии как **subtitle** ячейки **UITableView**. Если у фотографии нет заголовка, используйте описание (description) фотографии как **title** ячейки. Если у фотографии нет ни заголовка, ни описания, используйте “Unknown” как **title** ячейки **UITableView**. Ключи словаря с информацией о **Flickr** фотографии определены в **FlickrFetcher.h**

6. Когда пользователь выбирает фотографию из любого списка, показывайте ее

внутри прокручиваемого изображения (scrolling view), которое позволяем пользователю выполнять жесты **pan** и **pinch** в разумных пределах. Вы можете получить **URL** для Flickr фотографии, используя метод **URLForPhoto:format:** класса **FlickrFetcher**.

7. Убедитесь, что заголовок фотографии находится где-то на экране при показе изображения (image) фотографии пользователю.

8. Как только изображение (image) фотографии появляется на экране, масштаб его должен устанавливаться автоматически, чтобы показать как можно больше фотографии без дополнительных, неиспользованных пространств. Как только пользователь начнет увеличивать (zoom in) или уменьшать (zoom out) фотографию с помощью **pinching** жеста, вы должны прекратить автоматическое масштабирование изображения фотографии.

9. В вашем приложении main thread (основной поток) никогда не должен блокироваться (например, получение данных от Flickr должно выполняться в другом потоке).

10. Ваше приложение должно работать как в портретной, так и в ландшафтной ориентации, как на iPhone, так и на iPad. Используйте подходящие специальные UI идиомы (например , не позволяйте вашей iPad версии выглядеть на экране как гигантская iPhone версия).

11. Список “недавних” фотографий должен сохраняться в **NSUserDefaults** (то есть они должны постоянно сохранятся между запусками данного приложения). Будет более удобно, если массивы (arrays), которые возвращаются с **Flickr**, все были бы property lists (сразу же после преобразования из формата JSON).

12. На этой неделе ваше приложение должно работать на реальном устройстве.

Подсказки (Hints)

1. Поместите свой собственный ключ Flickr API в файл **FlickrAPIKey.h** иначе все ваши запросы работать не будут.

2. Возможно начать разрабатывать эту домашнюю работу с шаблона Tabbed Application (или даже с шаблона Master-Detail Application) и это приветствуется, то тем не менее более понятным будет начать как обычно с шаблона Single View Application и перетащить **UITabBarController**, который вам нужен и использовать элемент меню **Editor** -> **Embed in** если это нужно и делать **Ctrl**-перетаскиванием нужные установки Relationships и Segues. Важная часть этой домашней работы - усилить ваше понимание того, как работают различные механизмы storyboard вместе и как они относятся друг к другу.

3. Данные, возвращаемые с **Flickr**, представлены в JSON формате. iOS имеет встроенный JSON парсер. Просто создайте **NSData**, содержащие информацию, возвращаемую с **Flickr** (используйте метод **dataWithContentsOfURL:** класса **NSData**); затем преобразуйте JSON в property list (то есть объекты **NSArray** и **NSDictionary**) используя метод класса в **NSJSONSerialization** с именем **JSONObjectWithData:options:error:**. Вы можете передать 0 в качестве аргумента для **options**.

4. Первая вещь, которую вы, возможно, захотите сделать как только скопируете код **FlickrFletcher** к себе в приложение (и добавите свой API ключ) - это выберите данные с помощью **URLForTopPlaces** , затем сделаете парсинг JSON и наконец **NSLog()** результатов парсинга. Таким образом вы сможете увидеть формат выбранных данных **Flickr**. Тоже самое можно повторить, когда вы запрашиваете список фотографий для заданного места.

5. Верхний уровень результатов запроса **Flickr** - это словарь. Внутри этого словаря находится массив с вашими результатами. Например, для получения массива мест из данных, возвращенных методом **URLForTopPlaces** (давайте предположим, что вы уже преобразовали **Flickr** результаты из формата JSON в словарь с именем **results**) , вы, во-первых, можете использовать

NSDictionary *placesResults = results[@"places"] и тогда получаем массив **NSArray *places = placesResults[@"place"]**. Альтернативно, вы можете сделать это путем вовлечения одного метода:
NSArray *places = [results valueForKeyPath:@"places.place"];

6. В **FlickrFletcher.h** имеются **#defines** для всех интересующих вас ключей к данным из **Flickr**. У некоторых из них есть точки "." и их соответственно можно использовать только с **valueForKeyPath:**, например, **FLICKR_PHOTO_DESCRIPTION**.

7. Ключ **id** (в словаре, содержащем информацию о фотографии) является уникальным и постоянным идентификатором фотографии (он определен в **#define FLICKR_PHOTO_ID @"id"**).

8. Для создания MVC табличного вида, перетяните **TableViewController** из палитры объектов на ваш storyboard и измените его класс на пользовательский **subclass UITableViewController** (не забудьте установить в качестве superclass **UITableViewController** в диалоге **File** -> **New** -> **File** ->**Objective-C class** -> …).

9. Некоторые из вас еще достаточно нечетко понимают использование объектно-ориентированного программирования для инкапсуляции функциональности в вашем приложении. Например, допустим, что в вашем приложении вам понадобится 5 различных **UITableViewController subclasses** и вы могли бы подходящим образом "разделить" (share) код между ними, и получить прекрасно сконструированные, повторно используемые **Controller** с понятными **API** и **Model**. И это совершенно правильно: вначале создать **UITableViewController subclass**, который будет что-то делать, а затем создать **subclass** вновь полученного класса для того, чтобы делать уже что-то более изысканное.

10. В стиле того же "хорошего объектно-ориентированного дизайна" вы захотите собрать все ваши **NSUserDefaults** вызовы в отдельный вспомогательный класс вместо того, чтобы "рассеивать" различного формата "недавние" данные по многочисленным классам. Многие из вас, кто делали дополнительные задания (Extra Credit) на прошлой неделе, были очень

небрежны в этом.

11. Существуют прекрасные методы сортировки (особенно полезны те, что используют блоки) в **NSArray** и **NSMutableArray**. Обязательно познакомьтесь с ними.

12. Вам понадобится больше, чем массив **places** (мест) в качестве *внутренней* структуры данных для **TableViewController** , отображающей места, где сделаны **Flickr** фотографии, но можно использовать различные комбинации общих Foundation классов (таких, как **NSArray** и **NSDictionary**).

13. Каждый MVC должен быть снабжен информацией, в которой он нуждается прежде, чем он будет "вытолкнут" другим MVC, а затем уже будет функционировать по своему собственному сценарию. Другими словами MVC не должен зависеть от другого "выталкивающего" его MVC (за исключением подготовки, которую выполняет "выталкивающий" MVC в **prepareForSegue: sender:**).

14. Заметьте, что ячейки для всех **TableView** в этой домашней работе требуют **subtitles**, так что вы должны установить тип **Subtitle** для ячеек **TableView** в **Xcode** в качестве динамического прототипа.

15. Не забывайте, что идентификаторы для повторного использования (**reuse identifiers**) в **UITableViewCell**, установленные в **Xcode** для динамического прототипа ячеек должны совпадать с теми, что используются в методах **tableView:cellForRowAtIndexPath:**. Здесь может возникнуть небольшое недоразумение, если вы будете использовать **subclasses UITableViewController subclasses** (то есть подклассы подклассов UITableViewController) (так как вы затем наследуете **tableView:cellForRowAtIndexPath:**); поэтому выбирайте в качестве имен идентификаторов имена широкого назначения (которые кратко и в общих чертах описывают, что показывается в этой ячейке).

16. Оказывается, что **UIRefreshControl** не всегда появляется, если вы вызываете **beginRefreshing** программным путем. Если вы бесстрашны, то можете это доработать заставив **UITableView** подкручиваться вверх путем установки его **contentOffset** (помните, что **UITableView** является **UIScrollView**)

отрицательного значения координате y (установка ее равной высоте **height** **refreshControl** было бы лучшим вариантом). Однако эта доработка не является Обязательным заданием (старт **refreshControl** все равно происходит, даже если его не подкрутить чуть-чуть наверх, чтобы он появился).

17. Преобразовать **URL** изображения с Flickr в **UIImage** очень просто. Просто создайте **NSData** с содержимым этого **URL** (**[NSData imageWithContentOfURL:theURL]**), затем создайте **UIImage**, используя полученные **NSData** (**[UIImage imageWithData:imageData]**).

18. Увеличение/уменьшение масштаба **Scroll View** требует определенных вычислений, в которые вовлекаются границы (**bounds**) **UIScrollView** и размер (**size**) фотографии. Таким образом, вам нужно пересчитывать это всякий раз, как будут меняться границы **UIScrollView** или **frame** **UIImageView** внутри него. Из лекций не забывайте, где (в методах "жизненного цикла" **UIViewController**) рассчитывается геометрия **view** перед появлением на экране.

19. Вы можете очень просто получить заголовок для **Detail View Controller** в **SplitViewController** просто путем вставки **Detail View Controller** внутрь **UINavigationController**. Но для того, чтобы затем изменить его по своему усмотрению, нужно как-то добраться до **Detail View Controller**, и вы должны знать, как искать его в **UINavigationController** (то есть найти его **rootViewController**).

20. К тому времени, когда вызывается метод "жизненного цикла" **viewDidLoad**, методы делегата **UISplitViewControllerDelegate** уже были вызваны (особенно тот, который размещает **UIBarButtonItem** кнопку где-то на пользовательском интерфейсе **Detail View Controller** для возврата мастер **Master View Controller)**. Следовательно, вы должны установить вашего делегата **UISplitViewControllerDelegate** перед этим. И самое лучшее место для этого **awakeFromNib**.

21. Если вы хотите обновить **Detail View Controller** в **SplitViewController** на iPad из **Master View Controller**, который является **TableViewController**, возможно, вы захотите выполнить метод **tableView:didSelectRowAtIndexPath:** в **Master View Controller** (этот метод своего рода "target/action" метод для **TableView**), а не

segueing (“переезд”). В этом методе вы будете делать то же, что и в методе **prepareForSegue: sender:** (то есть устанавливать Model для **destinationViewController**).

22. Так как **View Controller** с изображением (**image**) фотографии на iPad все время находится на экране, ему необходимо правильно реагировать на изменение во времени **URL** изображения фотографии (например, на изображения, имеющие различные размеры).

23. Если вы переустанавливаете изображение (**image**) для своего MVC, (например, через **Detail View Controller** в **SplitViewController**), будьте внимательны и заново переустановите значение свойства **zoomScale** вашего **UIScrollViewController** в 1.0 перед тем, как переустановить **contentSize** **ScrollView** для нового изображения (**image**). Свойство **zoomScale** воздействует на **contentSize** (например, если увеличивается масштаб (zoom in), то **contentSize** автоматически настраивается так, чтобы быть больше, если масштаб уменьшается (zoom out), то становится **contentSize** меньше), так что если у вас **zoomScale** не равен 1.0 и вы начинаете изменять **contentSize**, то получите результаты, отличные от ожидаемых.

24. Метод **mutableCopy** массива **NSArray** может быть взят на вооружение, если вы хотите что-то добавить в структуру данных, которые уже сохранены неизменяемом (immutable) способом в **NSUserDefaults**.

25. Как всегда, количество кода, требуемое для реализации этого приложения, не такое громадное (оно может быть выполнено менее, чем 150 строками кода, если считать только то, что между фигурными скобками). Если вы чувствуете, что вам нужны дюжины (12) строк кода для выполнения какой-то одной возможности, то, возможно, существует какой-то другой, более лаконичный способ. Используйте объектно-ориентированный механизм повсюду для уменьшения объема кода.

26. Просто используйте текст, который следует за последней запятой в имени места в качестве имени страны этого места.

Что нужно изучить.

Это частичный список концепций, которые нужно использовать при разработке этого домашнего задания, демонстрируя при этом их знание.

1. **UITableView**.
2. Реакция на выбор в **UITableView** (как с помощью **segue**, так и **didSelectRowAtIndexPath:**).
3. **UIRefreshControl**.
4. **UIActivityIndicatorView**.
5. Многопоточность.
6. **NSURLSession**.
7. **UIScrollView**.
8. **UIImageView**.
9. **UISplitViewController**.
10. **NSUserDefaults**.
11. JSON Parser.
12. Еще больше транзита между MVC (**prepareForSegue:sender:**, и т.д.).

Оценка (Evaluation)

Во всех заданиях требуется написание качественного кода, на основе которого строится приложение без ошибок и предупреждений , следовательно вы должны тестировать полученное приложение до тех пор, пока оно не начнет функционировать правильно согласно поставленной задачи.

Приведем наиболее общие соображения, по которым задание может быть отклонено:

- Приложение не создается (Project does not build).
- Приложение не создается без предупреждений (Project does not build without warnings).
- Один или более пунктов в разделе **Обязательные задания (Required Tasks)** не выполнены.
- Не понята фундаментальная концепция задания.
- Код - небрежный или тяжелый для чтения (например, нет отступов и т.д.).
- Ваше решение тяжело (или невозможно) прочитать и понять из-за отсутствия комментариев, из-за плохого наименования методов и переменных, из-за непонятной структуры и т.д.
- Решение нарушает MVC.
- Пользовательский интерфейс (UI) в полном беспорядке. Элементы должны быть выравнены и надлежащим образом дистанцированы друг от друга, чтобы создать благоприятный вид UI. Xcode предоставляет в ваше распоряжение пунктирные голубые направляющие линии для этого, так что не может быть никакого прощения невыраненным линиям и т.д. Возьмите в привычку всегда создавать эстетически сбалансированный UI с самого начала этого курса.
- Задание было сдано поздно (у вас есть 3 свободных дня, так что используйте их мудро).
- Неверное или слабое использование объектно-ориентированных принципов конструирования. Например, код не следует дублировать, если вы можете его повторно использовать через наследование или другие объектно-ориентированные методологии конструирования.

Часто студенты спрашивают: “Сколько комментариев кода нужно сделать?” Ответ: ваш код должен легко и полностью быть понятным любому, кто его читает. Вы

можете предполагать, что читатель знает SDK, но вам *не следует* предполагать, что он уже знает решение проблемы.

Дополнительные задания (Extra Credit)

На этой неделе не будет дополнительных заданий.