Задание IV: Smashtag Mentions (Выборка данных из Twitter)

Цель:

Вы должны усовершенствовать приложение Smashtag, которое мы создали на лекции, чтобы обеспечить быстрый доступ к **hashtags**, **urls**, **images** и **users**, упомянутым в твите.

Обязательно посмотрите ниже раздел подсказок (**Hints**)!
Также проверьте последние изменения в разделе Оценка (**Evaluation)** чтобы убедиться, что вы понимаете, что будет оцениваться в этом Задании.

Материалы

- Это абсолютно новое приложение, так что вам ничего не нужно, кроме знаний, полученных из первых трех Заданий.
- Вам необходимо иметь Twitter account.
- Ряд вспомогательных классов для Twitter будет вам очень полезен!

Обязательные пункты задания

1. Усовершенствуйте приложение Smashtag, полученное на лекции, в плане выделения (разными цветами каждого) **hashtags**, **urls** и **user screen names**, упомянутых в тексте твита (они известны как "**mentions**"). Заметьте, что эти **mentions** уже обнаружены для вас в каждом твите и представлены как **[IndexedKeyword]**s в классе **Tweet** в поставляемом коде **Twitter**.

2. Когда пользователь кликает на твите, "переезжайте" (segue) на новый **UITableViewController**, у которого 4 секции, показывающие "**mentions**" в твите: Images, URLs, Hashtags и Users. Первая секция показывает (одно на

строку) любые изображения, прикрепленные к твиту (найдены в переменной **media** в классе **Tweet**). Последние 3 секции показывают элементы, описанные в Обязательном пункте №1 (опять, по одному на строку).

3. Каждая секция в таблице "**mentions**" должна иметь соответствующие заголовки.

4. Если в секции нет никаких элементов, то заголовок не должен быть виден для этой секции.

5. Если пользователь выбирает какой-то **hashtag** или **user** в таблице "**mentions**", созданной в вышеприведенном Обязательном пункте № 2, то вы должны куда-то "переехать" (segue), чтобы показать результаты поиска в Twitter этого **hashtag** или **user**. Это должен быть поиск именно **hashtags** или **users**, а не просто строки с **именем hashtag** или **user** (например, поиск "#stanford", а не "stanford"). **View Controller**, куда вы "переедите" (segue), должен работать точно также, как ваш главный **View Controller**, показывающий твиты (**TweetTableViewController**).

6. Если пользователь кликает на "меншене" **url** в вашем вновь созданном **View Controller**, вы должны открыть этот **url** в **Safari** (смотри раздел "Подсказки", приведенный ниже, и узнай как это сделать).

7. Если пользователь кликнет на изображении (**image**) в вашем вновь созданном **View Controller**, "переезжайте" на новый MVC, который позволит пользователю прокручивать (scroll) изображение и изменять его масштаб. Когда изображение впервые появляется в MVC, оно должно быть показано в увеличенном масштабе (но со своим нормальным соотношением сторон (aspect ratio)) и так, чтобы занять как можно больше экранного пространства без "белых зазоров" вокруг изображения.

8. Сохраняйте недавние 100 поисков в **Twitter**, которые пользователь выполнил в вашем приложении. Добавьте **UITabBarController** к вашему приложению с одной закладкой для поиска (то есть ваш главный UI) и второй закладкой, показывающей недавние поисковые термины, используемые для поиска в вашей таблице (они должны быть уникальны и первыми должны быть самые новые). Когда пользователь кликает на поисковом термине во второй закладке, "переезжайте" (segue) (оставаясь в той же самой закладке) куда-то, чтобы показать самые свежие твиты, соответствующие этому поисковому термину. Запомните эти недавно выполненные поиски в постоянном хранилище **NSUserDefaults**, так чтобы ваше приложение не забывало их в случае повторного старта приложения.

9. Сетевые запросы никогда не должны блокировать **main thread** в вашем приложении.

10. Ваше приложение должно работать правильно как в портретном, так и в ландшафтном режимах на любом iPhone (это приложение только для iPhone).

Подсказки (Hints)

1. Вам нужно залогиниться в **Twitter** в Установках (**Settings**) на вашем приборе (или на симуляторе), чтобы сделать работоспособными классы, поставляемые в папке **Twitter**.

2. Классы, поставляемые в папке **Twitter**, являются **Printable**, так что вы можете распечатать их с помощью **println**.

3. Пусть вас не шокирует код в классах **Twitter**. Единственный метод, который вам когда-либо придется вызвать из всего фреймворка - это **fetchTweets**. И еще вам понадобиться доступ к определенным свойствам в структурах данных **Tweet**, **MediaItem** и **User**.

4. У большинства **UIKit** классов (таких как **UILabel** и **UIButton**) есть свойство **attributedText**, которое позволяет вам устанавливать и получать их текст с использованием **NSAttributedString**.

5. Убедитесь, что вы “не разрушили” возможность, которая существует в текущей версии Smashtag, состоящая в том, что для показа твитов используется **preferred body** фонт (это позволяет делать текст в твитах меньшего или большего размера в зависимости от пожеланий пользователя, которые он выражает в Установках (**Settings**)).

6. Для добавления **UITableViewController** на ваш storyboard, просто вытяните его из Палитры Объектов и измените его класс на пользовательский (custom) **subclass** класса **UITableViewController**, который вы создадите с помощью **New File**.

7. У вашего нового MVC с “**mentions**” (и **images**) в каждой секции элементы различного типа. Хотя у вас может быть искушение обращаться с ними при помощи длинных **if-then** или **switch** предложений в **UITableViewDataSource** и навигационных методах, более понятный подход заключался бы в создании внутренней структуры данных для вашей **UITableViewController**, которая инкапсулирует данные в секциях (их похожесть и различия). Например, было бы замечательно, если бы **numberOfSectionsInTableView**, **numberOfRowsInSection** и **titleForHeaderInSection** состояли бы из одной

строки.

8. Фактически, любой метод, который имеет больше дюжины (12) строк кода может быть плохо читабельным (и иметь неудовлетворительный архитектурный подход).

9. Не забывайте о таких возможностях **Swift** как **enum**. Используйте **Swift** на полную возможность. Заимствуйте структуры данных, которые мы создали для **CalculatorBrain**. Это может дать некоторое вдохновение для этого Задания.

10. Как всегда, хорошо подумайте, что будет "**public (то есть не private) API**" для вашего нового **C**ontroller. Все, что возможно, сделайте **private**. Ваш **public API** - это то, что говорит оставшемуся приложению: "Вот как вы используете этот **C**ontroller." Никакие другие части вашего приложения не должны знать ничего о внутреннем устройстве вашего **C**ontroller. И ваш **C**ontroller должен всегда "делать правильные вещи (то есть то, для чего он создавался)", если другая часть вашего приложения использует **C**ontroller путем вызова его **public API**.

11. То же самое можно повторить для любого создаваемого **subclass** класса **UITableViewCell**. Или вообще любого создаваемого для этой цели класса!

12. Также подумайте о заголовках ваших MVCs (то есть что появится в **Navigation Bar** вашего **Navigation Controller** при показе этого MVC).

13. Если вы собираетесь использовать индексирование внутри **NSAttributedString**, вам нужно использовать свойство **nsrange** структуры **IndexedKeyword**, а не свойство **range** (так как индексация **NSAttributedString** использует индексацию основополагающей строки как **NSString**, а не как **String**).

14. Если у вас есть **NSURL** с именем **url**, вы можете открыть его в **Safari** следующим образом: **UIApplication.sharedApplication().openURL(url)**.

15. Когда вы кликаете на **user** или **hashtag** в вашем "**mentions**" MVC, вы можете "переехать" (segue) на **Table View Controller** "список твитов" с помощью либо нормального "**Show**" **segue** либо с помощью "**Unwind**" **segue**. Вам решать, какой из них приведет к созданию лучшего пользовательского интерфейса.

16. Вам точно понадобятся два различных **UITableViewCell** прототипа на вашей storyboard. Дайте им различные идентификаторы и берите из очереди (dequeue) подходящий прототип в методе **cellForRowAtIndexPath**.

17. Высота строки в вашем новом **C**ontroller не нуждается в "оценке" как высота строки в "списке твитов", потому что у вас очень мало строк и производительность не играет решающего значения. Следовательно, вам

захочется реализовать метод **heightForRowAtIndexPath** делегата **UITableViewDelegate**.□

18. Для строк, содержащих изображение (**image**), вам придется рассчитать подходящую высоту. Для других строк в вашей таблице вы можете позволить рассчитаться высоте строки автоматически (используя Autolayout) путем возврата **UITableViewAutomaticDimension** из метода **heightForRowAtIndexPath**.

19. Вы можете рассчитать соотношение сторон (**aspect ratio**) изображения (**image**) в твите, не прибегая к реальной выборки **image** из своего **url**. Смотри класс **MediaItem** в дополнительных **Twitter** классах.

20. Замечательной возможностью вашего приложения является (должно быть!) то, что если пользователь захочет увеличить немного масштаб изображения (image) в твите без того, чтобы кликать на нем и “переезжать” на MVC для подробного показа image, пользователь может просто перевернуть прибор в ландшафтный режим. Если все выполнено правильно, то вы получите эту возможность бесплатно (то есть не написав ни строчки кода).

21. Для обязательного пункта, в котором пользователь может кликнуть на изображении (**image**) чтобы начать перемещать его с помощью жеста **panning** и масштабировать в новом MVC, вы можете практически полностью использовать код из демонстрационного приложения **Cassini**. Однако вам придется добавить в **ImageViewController** автоматическую “подгонку” (**autozooming-to-fit**) изображения под полный размер экрана путем изменения масштаба.

22. Было бы очень круто сделать так, чтобы автоматическая “подгонка” (**autozooming-to-fit**) работала бы и при изменении геометрии топового **view** этого MVC до тех пор, пока пользователь не стал бы явно менять масштаб с помощью жеста **pinching** (есть метод делегата, который позволяет обнаружить явное изменение масштаба пользователем). Этот способ позволит выполнить автоматическую “подгонку” изображения при повороте прибора пользователем.

23. Возможно, хорошая идея - иметь глобальное “хранилище” для недавних поисковых терминов, так как вам все равно нужно запоминать их в **NSUserDefaults**. Почему бы не сделать этим хранилищем **NSUserDefaults**? Вы можете создать маленький класс или структуру для хранения и извлечения данных из **NSUserDefaults**, которую можно использовать всюду в вашем приложении.

24. **NSData(dataWithContentsOfURL:)** блокирует поток, в котором эта функция вызывается при работе с сетевым **url**. Так что вы не можете вызывать ее из **main thread**.

25. Вы не можете делать вызовы **UIKit** за пределами **main thread**. Будьте внимательны, чтобы “случайно” не сделать это вызовом некоторого метода, который впоследствии вызовет метод из UIKit. Если вы вызовите метод из UIKit (прямо или косвенно) за пределами **main thread**, ваш UI может аварийно завершится нередсказуемым образом.

26. Метод **fetchTweets** выполняет свой **handler** за пределами **main thread**.

27. Помните, что **cells** таблицы **UITableView** создаются только для видимых ячеек, и затем они используются повторно по мере того, как данные приходят на экран и уходят с экрана.

28. Если вы делаете выборку данных в потоке, отличном от **main thread**, затем получаете результат и хотите просить **main queue** что-то сделать с этими результатами, вы должны убедиться, что ничего не будет “изменяться” до тех пор, пока идет работа с сетью.

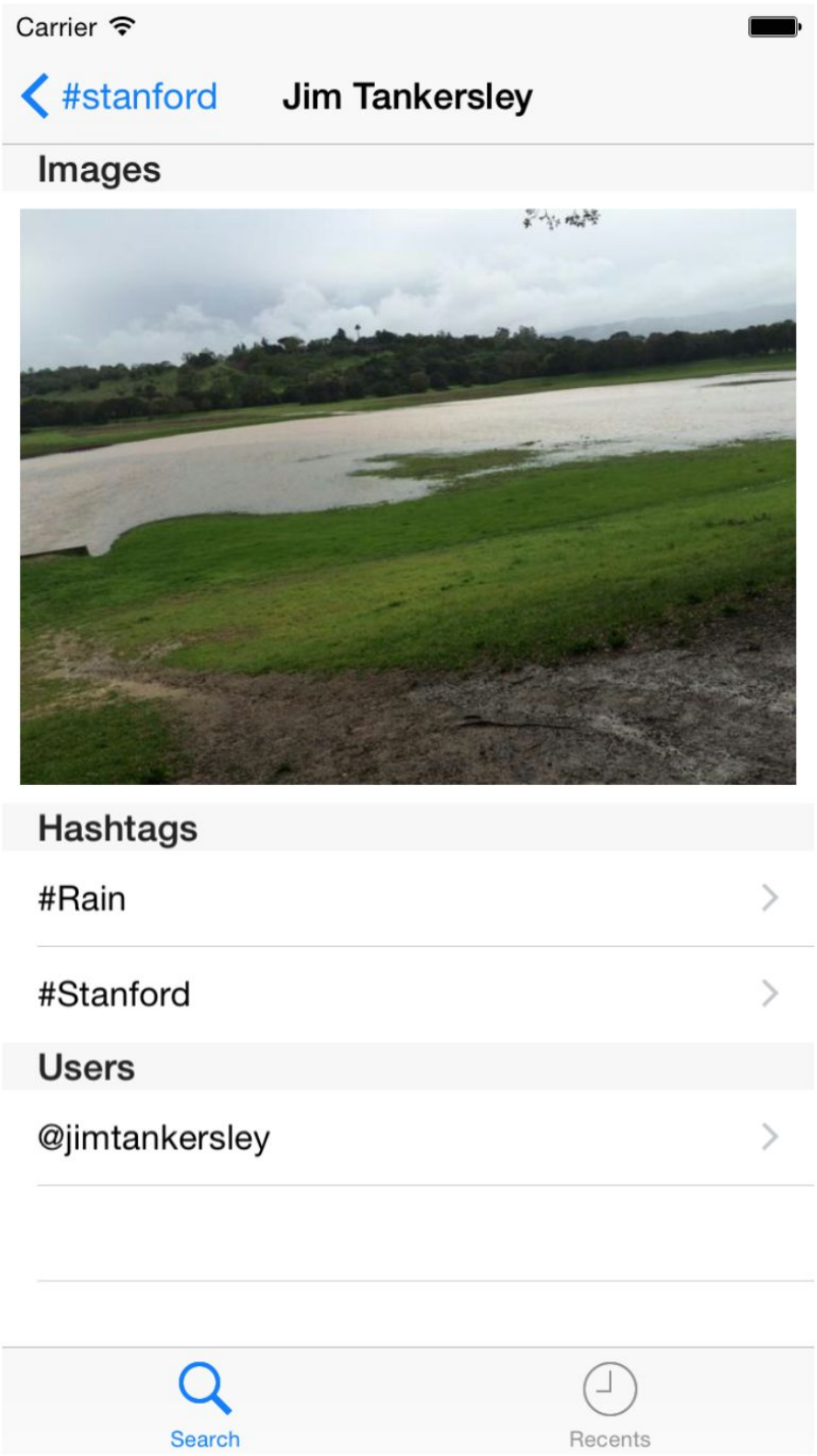
Что нужно изучать

Это частичный список концепций, в которых это задание увеличивает ваш опыт работы или продемонстрирует ваши знания.

- NSAttributedString
- UITableView
- UITableViewController
- UITableViewCell
- UIRefreshControl
- UIActivityIndicatorView
- UITabBarController
- Multithreading (многопоточность)
- Data structure design
- NSUserDefaults
- UIScrollView
- UIImageView

Screen Shots (Экраны приложения)

Мы всегда колеблемся, стоит ли включать примеры экранов пользовательского интерфейса, потому что мы не хотим ограничивать вашу креативность. Приведенные ниже примеры “экранов” приложения НЕ ЯВЛЯЮТСЯ Обязательными пунктами. Они просто дают вам идею, если у вас есть какие-то затруднения при визуализации Обязательных пунктов. Цвета выбраны абсолютно случайно. Вы должны сами подобрать цвета для своего приложения.


Carrier
#stanford
Jim Tankersley
Images
Hashtags
#Rain
#Stanford
Users
@jimtankersley
Search
Recents

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 warning)

- Один или более пунктов в разделе **Обязательные задания (Required Tasks)** не выполнены.
- Не понята фундаментальная концепция задания (A fundamental concept was not understood).
- 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.
- UI в полном беспорядке. Элементы UI должны быть выравнены и размещены с зазором для того, чтобы хорошо выглядеть.
- Неверное или плохое использование принципов объектно-ориентированного конструирования. Например, код не должен дублироваться, если его можно повторно использовать путем наследования или другой методологии объектно-ориентированного конструирования.
- Public и private API разграничены неправильно.

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)

Ниже приводится большое количество идей. Мы определенно не ожидаем, что вы выполните все из них (некоторые значительно труднее остальных). Прочтите все и выберите те, которые вас заинтересовали больше всего.

1. В секции **Users** вашего нового **UITableViewController** представьте список не только пользователей, **users**, упомянутых в твите, но также и пользователей **users**, которые послали твит в первое место.

2. Когда вы кликаете на **user** в секции **Users**, ищите не только те твиты, в которых “упоминается” этот **user**, но и твиты, где это пользователь **user** посылает твиты.

3. Если вы “переезжаете” (segue), используя **Show** (а не **Unwind**), добавьте какой-нибудь UI элемент, который всегда будет возвращать вас (**Unwind**) к **rootViewController** в **UINavigationController**. Даже если вы используете **Unwind** (а не **Show**), то если вы выполняете дополнительный пункт, связанный с **Collection View**, используя **Show** segue, возможно, вы захотите реализовать “unwind to root” поведение в экранных фрагментах (scenes), на которые вы “переезжаете” через **Collection View**.

4. Вместо открытия **urls** в **Safari**, покажите их в вашем приложении путем “переезда” (segueing) на **C**ontroller с **UIWebView**. Вам понадобиться обеспечить по крайней мере простейшее “управление браузером” с помощью элементов UI, которые приходят вместе с **UIWebView** (например кнопка “назад” (“back button”)).

5. Сделайте таблицу с “недавними поисковыми терминами” редактируемой (то есть позвольте пользователю использовать жест **Swipe влево** для удаления терминов, которые ему не нравятся).

6. Добавьте некоторый UI элемент, который показывает новый **View Controller**, отображающий **UICollectionView** всех первых изображений (**image**) (или, если хотите, всех **images**) во всех твитах, которые удовлетворяют условиям поиска. Когда пользователь кликает на изображении (image) в этом **UICollectionView**, “переезжайте” (segue) на показ этого твита.

Дополнительные задания (Подсказки)

1. Если вы создали хорошую внутреннюю структуру данных для данных секции, то этот дополнительный пункт сведется лишь к усовершенствованию метода **init()** для этой внутренней структуры данных.

2. Вам нужно познакомиться с “operators” в Twitter search queries.

3. Вы можете находится в ситуации, когда иногда вы хотите, чтобы появилась кнопка “unwind to root”(вернуться в начало) , а иногда - нет (например, если вы уже находитесь в корневом **View Controller**, то вам, очевидно, эта кнопка не нужна). Вы можете попытаться управлять этой ситуацией в коде (с помощью **if then** предложений вы можете показывать или скрывать кнопки “unwind to root”), или вы могли бы использовать полиформизм (polymorphism) и иметь в распоряжении корневой экранный фрагмент без кнопки “unwind to root”, при этом его **C**ontroller реализует “unwind to root” @IBAction метод, который выполнил бы unwind для “unwind to root” кнопок других экранных фрагментов (это не потребовало бы никакой реализации за исключением метода “unwind to root” в **subclass** non-root версии MVC). Другие экранные фрагменты (scenes) на storyboard имели бы кнопку, но не реализовали бы этот метод (потому что они используют **superclass** в качестве своего класса экранного фрагмента, а не **subclass**). Как часть этой технологии, возможно, вы захотите выполнять copy/paste целых экранных фрагментов (и затем менять только их класс). Все это вы могли бы делать исключительно с unwinds. Это просто пища для размышления. Если вас что-то смущает, то игнорируйте подсказку и идите своим путем.

4. Посмотрите документацию для **UIWebView**.

5. Когда вы создаете subclass **UITableViewController**, шаблон включает некоторые методы, чтобы помочь вам.

6. Мы должны рассмотреть несколько вещей…
   a. Шаблон, который вы получите при создании **subclass** **UICollectionViewController**, вызывает **registerClass** в **viewDidLoad**. УДАЛИТЕ ЭТУ СТРОКУ КОДА. Вместо этого вы будете устанавливать класс **UICollectionViewCells** на storyboard. Если вы не уничтожите этот вызов **registerClass**, то он переопределит все, что вы задали на storyboard, так как **viewDidLoad** вызывается после того, как выполнена загрузка со storyboard.
   b. Так как, очевидно, все ваши изображения будут загружаться за пределами **main thread**, то прокручивание (scrolling around) будет происходить

быстро, но, если вы будете повторно подгружать их снова и снова по мере того, как пользователь осуществляет прокрутку, то вы будете получать много пустого пространства, которое заполнится со временем, но это не будет выглядеть так уж здорово. Так что нужно кэшировать изображения (images). Познакомьтесь с классом **NSCache**. Это как **NSDictionary** (**objectForKey** и **setObject:forKey**), но добавляется концепция "стоимости" того, что будет в кэше, с помощью **setObject:forKey:cost:**. "Стоимостью" изображения мог бы быть размер в kb, например. **NSCache** будет выбрасывать из кэша элементы в любое время, когда ему захочется, так что вы всегда будете искать нужный NSURL (чтобы найти соответствующее изображение **UIImage**), использовать его, если вы нашли, или опять загрузить его, если вы не нашли. Вы захотите ассоциировать кэш с вашим **subclass UICollectionViewController**.

c. Большая разница между **UITableView** и **UICollectionView** состоит в том, что **таблица** (Table View) всегда имеет одно и тоже расположение элементов (то есть строки в единственном столбце). А **коллекция изображений** (Collection View) имеет свойство **UICollectionViewLayout**, которое определяет как располагаются ячейки (и поэтому является чрезвычайно гибким инструментом). **UICollectionViews** по умолчанию имеет тип расположения ячеек **UICollectionViewFlowLayout**, который похож на расположение символов в "выровненном тексте". Это прекрасно подходит для наших целей! Такие вещи, как размер ячейки, определяются делегатом как в Table View, так и в Collection View, но в Collection View делегат предоставляет протокол, который специально разработан для "движка" расположения ячеек. Для расположения ячеек типа **FlowLayout**, протокол называется **UICollectionViewDelegateFlowLayout**. Поэтому, если вы хотите управлять, например, размером ячеек, то вам надо реализовать метод **collectionView:layout:sizeForItemAtIndexPath:** протокола в вашем **subclass UICollectionViewController**.

d. Вы можете пойти более легким путем и выбрать предопределенный размер ячеек (predetermined size) в **UICollectionView** или, возможно лучше, выбрать предопределенную "область" (то есть **width** x **height**) для каждой ячейки (но выполнять настройку **aspect ratio** каждого изображения).

e. Было бы здорово выполнять жест "**pinching**" на **UICollectionView**, который заставит ячейки становится больше или меньше (то есть показывать больше или меньше изображений). Жест "**pinching**" можно выполнить тривиально просто, если будете следовать изложенному выше подходу к определению размера ячеек ("**pinching**" просто бы масштабировал область как увеличивая ее, так и уменьшая).

f. Для показа твита, на изображение которого вы кликнули, вы можете повторно использовать "список твитов", который следует модифицировать для показа определенного твита (твитов) (в дополнение к тому, что он уже умеет выбирать и показывать выбранные твиты).