

Assignment V:

Smashtag Mention Popularity

Objective

In this assignment, you will enhance the Smashtag application even further to do some analysis on all of the mentions in a search result using Core Data.

Submit your solution via the normal process before the start of lecture on Monday.


Be sure to review the Hints section below!

Also, check out the latest in the Evaluation section to make sure you understand what you are going to be evaluated on with this assignment.

Materials

- You will need to have successfully completed the previous assignment to do this assignment.
 - You will probably want to use the [CoreDataTableViewController class](#) from the lecture demo.
-

Required Tasks

1. In your Recent Searches tab, add a Detail Disclosure button  to every row. Touching it must segue to a new table view MVC which lists all user and hashtag mentions (uniqued, case-insensitively) in all tweets ever fetched for the search term in that row.
2. This mentions popularity table should be sorted in order from the most popular mention in all tweets returned by that search term to the least popular.
3. If two (or more) mentions have the same popularity, those mentions should appear in the table in alphabetical order.
4. Each row in the table should show not only the mention, but also the number of times that mention was mentioned in tweets found by the given search term.
5. Do not put any mentions that were only ever mentioned once into the table. If a newly fetched tweet includes such a mention, though, the mention must immediately start appearing in the table (because then it would have been mentioned more than once).
6. Once the mentions in a tweet have been counted, they should not be counted again if the same tweet is fetched again. In other words, don't "double-count" mentions in tweets that are fetched more than once.
7. All of the data which drives this new MVC must be stored in Core Data (and persist across launchings of your application).
8. The contents of this new MVC must be managed by an `NSFetchedResultsController` (you may use `CoreDataTableViewController` as your new MVC's Controller's superclass if you wish).
9. Don't break any of the rest of the functionality in your Smashtag application. There is no requirement that the MVCs you already have from the previous assignment be converted to use Core Data, only your new MVC. Of course you'll need to modify your existing code to *collect* the data this new MVC is going to use.
10. You must not block the main thread of your application at any time. You can assume that any Core Data calls you make will not take so long that your main thread will be blocked (hopefully you design your schema such that this is actually true, even for very large numbers of downloaded tweets).
11. Your application must work properly in portrait and landscape on any iPhone (this is an iPhone-only application).
12. You must get this assignment working on an actual iOS device (not just the simulator).

Hints

1. You can get the `managedObjectContext` you need either from a `UIManagedDocument` (which is an Extra Credit item) or from the `AppDelegate` (by creating a new project and copying/pasting the appropriate code from `AppDelegate.swift`).
2. If you copy/paste from the `AppDelegate`, be sure the name of your CoreData model file is properly set in that code.
3. A way to check that you have satisfied Required Task 6 is to search for a term, check the mention popularity, then search for it again (not refreshing it, but typing it in again) and making sure that the popularities don't change (or at least, only change by newly-tweeted tweets). If the popularity counts double, you know it's not working.
4. Remember that to make `CoreDataTableView` do its thing, you simply need to set its `fetchedResultsController` property. The key is to create that `NSFetchedResultsController` with the correct `NSFetchRequest` (and `NSSortDescriptors`).
5. You are not required to display the full contents of Tweets or Users in your new MVC, so there's no need to build a huge schema that stores all the data that is fetched from Twitter. Just store the stuff you need to make your new MVC work.
6. With the proper schema, this entire application can be built with very straightforward sort descriptors and predicates. **Put your brainpower into designing the right schema rather than trying to build complicated predicates.**
7. Don't limit your idea of what the entities and attributes in your database schema need to be to what the properties on the Twitter framework classes are (even though we did exactly that in the simple lecture demo).
8. You are very likely to want an attribute somewhere in your schema which is an integer.
9. All `NSManagedObject` instances know the `NSManagedObjectContext` they are part of (they have a `var` which returns it). You will likely find this useful. It will keep you from constantly having to go back to your `UIManagedDocument` or `AppDelegate` to get the managed object context.
10. You'll recall from lecture that to-many relationships in the database are `NSSet`s in your code. `NSSet` is not mutable (i.e. you can't add anything to or remove anything from one). If you want to add or remove something from a to-many relationship, you can use the method `mutableSetValueForKey(String)`. This takes the name of the to-many relationship as an argument and returns you an `NSMutableSet` that you can add or remove objects from to update the database.
11. Remember that Core Data automatically updates the inverse of any relationship. So if you add something to a to-many relationship `NSSet`, it will automatically make the other side the the relationship correct (even if that other side is also a to-many relationship too).

12. You are welcome to use the code from the in-class demo as a guide, but you will learn much less by copying/pasting code (and then modifying it) than you will by typing it in from scratch.
13. You can distinguish one Tweet from another using the `id` field in `Twitter.Tweet`. In other words, that `id` is unique across all Tweets in the universe. If you are trying to make this work in Objective-C, it is strongly advised that you NOT have an attribute named `id` in your schema!
14. Don't forget to explicitly `save()` your managed object context if you are using the `AppDelegate` approach. If you are using the `UIManagedDocument` approach, you can force it to autosave by hitting the home button on your device or simulator before hitting the stop button in Xcode.
15. Always use `performBlock` or `performBlockAndWait` to do anything in a managed object context.

Things to Learn

Here is a partial list of concepts this assignment is intended to let you gain practice with or otherwise demonstrate your knowledge of.

1. Core Data
2. Application Delegate
3. NSManagedObjectContext
4. NSFetchedResultsController
5. Detail Disclosure Segues
6. Database schema design
7. UIManagedDocument (EC)

Evaluation

In all of the assignments this quarter, writing quality code that builds without warnings or errors, and then testing the resulting application and iterating until it functions properly is the goal.

Here are the most common reasons assignments are marked down:

- Project does not build.
- Project does not build without warnings.
- One or more items in the Required Tasks section was not satisfied.
- A fundamental concept was not understood.
- Code is visually sloppy and hard to read (e.g. indentation is not consistent, etc.).
- Your solution is difficult (or impossible) for someone reading the code to understand due to lack of comments, poor variable/method names, poor solution structure, long methods, etc.
- UI is a mess. Things should be lined up and appropriately spaced to “look nice.”
- Incorrect or poor use of object-oriented design principles. For example, code should not be duplicated if it can be reused via inheritance or other object-oriented design methodologies.
- Public and private API is not properly delineated.

Often students ask “how much commenting of my code do I need to do?” The answer is that your code must be easily and completely understandable by anyone reading it. You can assume that the reader knows the SDK, but should not assume that they already know the (or a) solution to the problem.

Extra Credit

There are lots of ideas below. We certainly don't expect that you'll do all of them (and some are more difficult than others). Read through them and pick whichever ones intrigue you the most.

1. Use `UIDocument` to store all your Core Data information. Be careful to get the asynchrony right.
2. Divide your mentions popularity table into two sections: Hashtags and Users. Again, with the right database schema, this can be implemented in very few (maybe two?) lines of code.
3. Loading up a lot of data by querying for an existing instance in the database, then inserting if not found over and over again, one by one (like we did in lecture), can be pretty poor performing. Enhance your application to make this more efficient by checking for the existence of a pile of things you want to be in the database in one query (then only creating the ones that don't exist). The predicate operator `IN` might be of value here.
4. You are not required to ever delete anything from your database, however, it only needs information about the most recent searches so, over time, there's wasted space in there. Have your application prune the database of objects that are no longer of interest (i.e. can't be accessed from the UI) to keep it a manageable size. It's up to you to decide when is a good time to do such pruning.