Assignment V:

Image Gallery

Objective

The goal of this assignment is to understand Table View, Collection View, multithreading, Scroll View and Text Fields.

Start this assignment from scratch. It is unrelated to the first four assignments of the quarter.

This assignment must be submitted using the submit script described here by the start of lecture next Wednesday (i.e. before lecture 14). You may submit it multiple times if you wish. Only the last submission before the deadline will be counted.

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 want at least the Utilities.swift file (you'll need the `imageURL` var which is added to `URL` via an extension) out of the EmojiArt demo code from this week. You're welcome to use other code from all the code posted this week.

Required Tasks

1. Build an application centered on a Collection View which contains images dragged in via Drag and Drop. A collection of such images is called an "Image Gallery."
2. Only allow dropping of items which have both a `UIImage` representation and a `URL` representation.
3. Use the `UIImage` in the drop for one purpose only: to determine the aspect ratio with which to display the image you fetch from the `URL` (i.e. do not use the dropped `UIImage` to actually draw anything).
4. Every item in the Collection View should have exactly the same width, but each item's height should be determined by its aspect ratio to that width.
5. The user must also be able to rearrange the items in the Collection View via Drag and Drop.
6. Never block the main thread. Fetching `URL`s must be done off the main queue.
7. Any time that an image is being fetched into a cell from its `URL`, an activity indicator should be spinning in that cell letting the user know that you are working on it.
8. Don't cache images. Fetch them from their `URL` each time you need them.
9. Implement a pinch gesture on your entire Collection View which scales the width of your cells (remember, all cells have the same width).
10. Add a Split View Controller to your application whose Detail is the Collection View above and whose Master is a Table View which lets the user choose an Image Gallery by name (i.e. this is a Table View full of the names of Image Galleries that, when touched on, opens up an Image Gallery in the Collection View described above, showing all of its images and allowing the user to drop more in).
11. Implement swipe to delete Image Galleries, however, deleting them just moves them to another section in your table called "Recently Deleted" (so your Table View will have two sections, one with no title, and one with the title Recently Deleted). Deleting an Image Gallery from the Recently Deleted section permanently deletes it from the table.
12. Implement swipe (in the other direction) to undelete an Image Gallery from the Recently Deleted section (i.e. move it back to the other section). Do not allow the user to open a Recently Deleted Image Gallery without undeleting it first. See the Hints for how to do this sort of "swiping in the other direction" UI.
13. Allow users to double tap on an Image Gallery in your Table View to start editing its name via a `UITextField` in the row in the table (i.e. edit the name in place).
14. When a user taps on a cell in your Collection View, segue to a new MVC which presents the image in a scroll view that fills the entire MVC so that the user can zoom

in and out to examine the image in detail. This means your Collection View will be embedded inside a Navigation Controller.

15. Your Image Galleries do NOT have to persist between runnings of the application. We'll be doing that next week.

16. In order to simplify this assignment (and let you focus on the primary things to learn), it is allowed for your Table View to have no selection even if you are editing an Image Gallery in your Collection View (this might happen, for example, at startup or because you deleted the Image Gallery that is currently showing in the Collection View or you changed the name of or undeleted some other document and now nothing is left selected). But when the user touches on a row in the Table View, it should open that Image Gallery up and start editing it. That's the requirement.

17. This is an iPad-only application (this week anyway).

Hints

1. You will NOT want to use the `ImageFetcher` object from lecture in this assignment. Just fetch the `URL` like Cassini did. That's because `ImageFetcher` relies on a backup image if the `URL` fails to load and this assignment only uses the `UIImage` to get the aspect ratio (i.e. it is not allowed to use it as a backup).
2. VERY IMPORTANT: The `URL`s that come from places like Google actually need to be massaged a little bit to get at the pure image `URL`. The `Utilities` provided with the EmojiArt demo have a simple extension to `URL` to accomplish this called `imageURL`. Essentially, every time you go off to fetch an image, do it with the `URL`'s `imageURL`. If you try to fetch the `URL` directly, it will likely not work much of the time.
3. When a drop happens, you'll have to collect both the aspect ratio (from the `UIImage`) and the `URL` before you can add an item. You could do this straightforwardly with a couple of local variables that are captured by the closures used to load up the drag and drop data.
4. The best implementation of dropping into your Collection View would use a placeholder (as shown in lecture), but that is not required by the Required Tasks. In some ways it might be easier to do with placeholders.
5. Remember that your `UICollectionViewCell` subclass(es) can have logic in them too. They are not limited to being just a container for outlets.
6. It should be quite obvious if you've gotten the multithreading right because if you drop a bunch of large images in and then zoom in so that as you scroll around enough that you are reusing cells a lot, you should see a bunch of spinning wheels appearing. That's what we'll be doing when we grade it!
7. When you change the width of the cells in your `UICollectionView`, you'll need it to lay itself out again. The most efficient way to do this is to directly ask the `UICollectionView`'s `UICollectionViewFlowLayout` to do its thing. You can do this by invalidating the layout of the `UICollectionViewFlowLayout`. If you define this convenience `var` in your Controller …

```
var flowLayout: UICollectionViewFlowLayout? {
    return collectionView?.collectionViewLayout as? UICollectionViewFlowLayout
}
```

   … then you can invalidate the flow layout with this line of code …

```
flowLayout?.invalidateLayout()
```

   … and the `UICollectionView` will immediately re-layout the cell positions using your new `itemSize` (which you are likely providing via the delegate method mentioned in lecture).
8. The way you do undelete swipes is with the `UITableViewController` method `leadingSwipeActionsConfigurationForRowAt:`. This method returns a

`UISwipeActionsConfiguration` that contains a list of `UIContextualActions`. All you need to do is create one of these `UIContextualActions` for Undelete which takes a closure that does the undelete. Remember to modify both your Model and the Table in that closure.

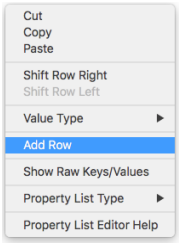
9. You will not need to do the "swap in a different cell" trick we did in EmojiArt to do your Image Gallery name editing in your Table View. Simply use a `UITextField` all the time to draw the name and set `isEnabled` on it to turn editability on when needed based on the double tap gesture.
10. You can prevent segueing from the Recently Deleted rows either by using a different prototype for them, or by implementing `shouldPerformSegue(withIdentifier:)`.
11. When you delete the last row in a section (e.g. you undelete the last Recently Deleted Image Gallery), you might have to send `reloadSections` to the table view for that section to get it to clear out the section title for that now-empty section. It seems like `UITableView` should do this automatically, but in some circumstances, it doesn't.
12. When you update your Model, you can often just `reloadData` to update it, but this will not be animated and be sort of jerky. Using `deleteRows`, `insertRows`, `moveRows`, etc. will provide animation and look a lot better (but is not Required).
13. Any time you modify your table with a combination of methods like `deleteRows` or `insertRows` or `moveRows` you have to do all the modifications together or you'll crash because your Table View's UI will be temporarily out of sync with your Model. You group calls to these methods together with the `performBatchUpdates` method. This also nicely has a `completion` handler that you can use to do something after the adjustment to the Table View is finished if you need to.
14. If the result of your fetch of a URL is not a valid image, you'll probably want to put some indication of that in the resultant Collection View cell. Maybe a frowny face or a note to that effect for the user? Just having a blank space in your Collection View might be a bit confusing to the user. Up to you.
15. One of the trickier parts of this assignment, actually, is the test of your MVC skills that comes along with being able to segue from your Table View MVC to your Collection View MVC. The Model of your Collection View MVC is essentially an Image Gallery. The Model of your Table View MVC is a list of those Image Galleries. The only trickiness is that both MVCs can edit an Image Gallery (because the Table View MVC can change the Image Gallery's name). Just be sure to come up with a simple data structure that allows your Table View MVC to hand an Image Gallery off to your Collection View MVC and still retain the ability to change the Image Gallery's name.
16. Your internal data structure for an Image Gallery is pretty simple (it just needs to have a list of `URL`s and their aspect ratios). Don't overthink this part of the assignment.

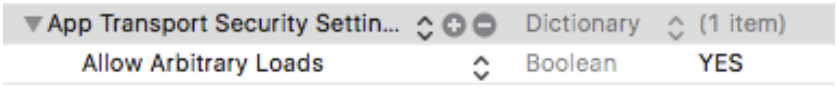
17. You might want to set your `splitViewController`'s `preferredDisplayMode` to `.primaryOverlay`. This is because the focus of your application is the Collection View and you don't want the Table View to be on screen very much (just when you need to switch to a different Image Gallery). Since this can get reset by iOS when your layout changes, the best place to set it is in `viewWillLayoutSubviews`, but be careful to check if it's already set because setting it again might cause a layout to happen and you'll get into an infinite loop! Also, you can add a button to the navigation bar of your Detail which will cause the Master to appear (i.e. do the same thing as the user swiping from the left) using this code in your Detail's `viewDidLoad` …

```
navigationItem.leftBarButtonItem = splitViewController?.displayModeButtonItem
```

18. It seems that sometimes when you drag into a Collection View and it starts rearranging the items to make room for a potential drop, that if you drag off the screen entirely, the Collection View will get into a bad state where it is "mid-reordering". This usually leads to a crash later (with complaints about reordering while already in the process of reordering), which you can ignore. Watch the course message boards for more on this.

19. If you drag in a URL that is insecure (i.e. it's `http://` instead of `https://`), then it will not be accepted by `UIImage` by default. You can change this in your `Info.plist` file. Click on `Info.plist`, then right click on its background and choose `Add Row` from the context menu that comes up, then scroll up to choose `App Transport Security Settings` and then hit the little triangle to the left of the new row that is addd to make the triangle point down, then hit the + button to add a sub row, then choose `Allow Arbitrary Loads` and set it to `YES`. After doing this, you should have an entry like this in your `Info.plist` …





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. Drag and Drop
2. Collection View
3. Table View
4. Text Field
5. Scroll View
6. Multithreading
7. Delegation

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.
- One or more items in the Required Tasks section was not satisfied.
- A fundamental concept was not understood.
- Project does not build without warnings.
- Code is visually sloppy and hard to read (e.g. indentation is not consistent, etc.).
- Violates MVC.
- UI is a mess. Things should be lined up and appropriately spaced to "look nice."
- Improper object-oriented design including proper use of value types versus reference types.
- Improper access control (i.e. `private` not used appropriately).
- 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.

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 iOS API, but should not assume that they already know your (or any) solution to the assignment.

Extra Credit

We try to make Extra Credit be opportunities to expand on what you've learned this week. Attempting at least some of these each week is highly recommended to get the most out of this course. How much Extra Credit you earn depends on the scope of the item in question.

If you choose to tackle an Extra Credit item, mark it in your code with comments so your grader can find it.

1. Let users drag things out of your Collection View into a trash can (maybe in the navigation bar at the top) which will delete the `URL` from the Image Gallery.
2. Make your Image Galleries persist between launchings of your application using `UserDefaults`. We're going to learn about persistence next week, but we probably won't be using `UserDefaults`, so this is a good opportunity to learn about that.
3. Keep your Table View in sync with your Collection View at all times. This can be a bit trickier than it might seem. Essentially, you must **always** have a selection in the Table View and that selected row must always be what is showing in the Collection View. Table View can be quite persnickety about selecting rows with `selectRow(at:)`. If it's finishing off an animation, for example, it might not select. You might even find you have to use a `Timer` to delay the selection a little bit to get it to take. Of course you'll also have to `performSegue` when you select a row on the user's behalf. There's also the issue of selecting a row when the Table View first appears. You'll likely have to wait until at least `viewDidAppear:` to do that.