## Requirements:

Modify the provided Assignment5 project to use CoreData as described below.

**Model Entities**: Two CoreData entities should be created in the CoreData model file: `Category` and `Image`. The `Category` entity should have a required indexed attribute of type `String` to store the category title and an optional to-many relationship to `Image`. The `Image` entity should have a required attribute of type `String` to store the image name, a required indexed attribute of type `Integer 32` to store an ordering value, and a to-one required relationship to `Category` (make sure this relationship is setup as the inverse of the one on the `Category` entity).

Custom subclasses of `NSManagedObject` should be created for these entities. To do this, open the data model file and for each entity in the data model ensure that the Codegen setting in the Data Model Inspector is set to "Category/Extension" and that the Module is set to "Current Product Module". Once this is done create the class files in the project's Source/Model folder with the "Create NSManagedObject Subclass…" option in the "Editor" menu. Xcode will generate 2 files for each entity you select in the create dialog. Once it has generated these files, delete the files with the "+CoreDataProperties" suffix and rename the other files so they have no suffix (i.e., instead of Category+CoreDataClass.swift, rename the file Category.swift).

These two remaining files are where you should place custom code for your entity classes (you can ignore/delete the comment at the top about automatic generation). In the `Category` class you should implement a computed property that returns the subtitle for the category. The `Image` class will not have any additional functionality.

**CatService**: The `CatService` implementation should be modified such that it no longer uses the plist based cat data content except to initial the CoreData store if necessary, and instead relies on CoreData. For the symbols in CoreData to be available in your code, you will need to import CoreData at the top of your file.

The `init()` initializer creates an `NSPersistentContainer` and loads its persistent stores. In the completion handler for this there is a TODO comment where you should access the `NSManagedObjectContext` provided by the container and use its `perform` method to convert the plist cat data into the instances of the entities you created above and save them (hint: It's important to call the `save()` method on the context when you're done making changes). You should use a guard statement so that you only create new instances of the cat data if they do not already exist in Core Data.

The `catCategories()` method should should be reimplemented to create an `NSFetchRequest<Category>` instance setup to sort by the title attribute. It should then return an `NSFetchedResultsController<Category>` for that fetch request.

The `imageNamesForCategory(atIndex:)` methods should be renamed to `images(for:)` and take a parameter of type `Category`. It should be reimplemented to create an `NSFetchRequest<Image>` instance setup to sort by the ordering value attribute and filter the returned images to only ones related to the category parameter. It should then return an `NSFetchedResultsController<Image>` for that fetch request.

Hint: It's important to call the `performFetch()` method on your `NSFetchedResultsController` instances. If you do not it will never provide any results.

**View Controllers:** Both view controller implementations should be modified to utilize the appropriate `NSFetchedResultsController` instance returned by the `CatService` instead of the previous data. You will also need to modify the value being set on the `CatImagesViewController` to be the selected `Category` instance instead of an index. It is not a good idea to create an entirely new fetched results controller every time you need to access data in your view controller. To avoid this, you should create an instance variable in your view controllers to store a reference to the fetched results controller instead and just request it as the view is loading. A good place to do this is in the `viewDidLoad()` method (make sure to call `super.viewDidLoad()`).

One of the benefits of `NSFetchedResultsController` is that you can conform to its delegate protocol to be informed of changes to the data is providing. This is extremely useful in situations where the data can change dynamically (for example, do to a background refresh process that gets new data from a web service).

One problem that commonly occurs when dealing with data is the situation where the data is not available as the screen is presented. There is an example of this in our application: if you delete the app from the simulator the data will not show the first time the app is run. This is because the initial data creation is done asynchronously and happens after the first screen is visible. To fix this small bug modify your view controllers to conform to the `NSFetchedResultsControllerDelegate` protocol. The methods of this protocol give detailed notifications about changes to the data. For this application we do not need to use the more detailed methods however. Implement the `controllerDidChangeContent(_:)` method and call the `reloadData()` method on your table view or collection view. Don't forget to set the delegate property of the fetched results controller after you retrieve it from the service.