





No description, website, or topics provided.

 3 commits

 4 branches

 0 releases

 1 contributor

Branch: day2 ▾

View #9



Create new file


Upload files

Find file


Clone or download ▾

This branch is 2 commits ahead of master.

 #99  Compare

 **SpencerCurtis** Update README.md to include correct screen recording

Latest commit f3db496 on Aug 14, 2018

 [README.md](#)

Update README.md to include correct screen recording

5 months ago

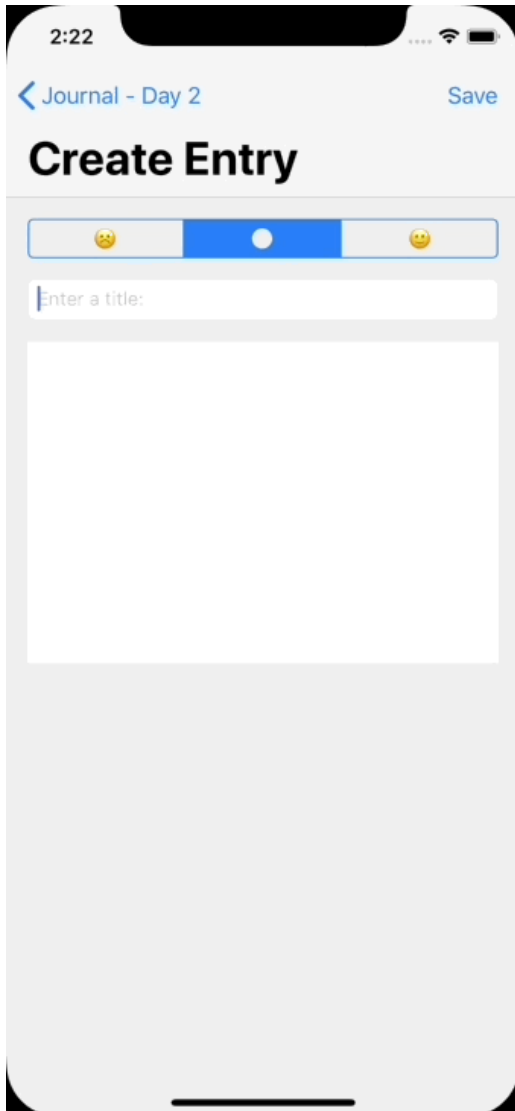
 [README.md](#)

# Journal (Core Data) Day 2

## Introduction

Today you will take the Journal project you made yesterday and add more functionality to it. This will help you practice migrating data in Core Data and `NSFetchedResultsController` .

Please look at the screen recording below to know what the finished project should look like:



## Instructions

Use the Journal project you made yesterday. Create a new branch called `day2`. When you finish today's instructions and go to make a pull request, be sure to select the original repository's `day2` branch as the base branch, and your own `day2` branch as the compare branch.

### Part 1 - Mood Feature

#### Segmented Control Addition

In order to add the functionality seen in the screen recording, which is the ability to set your mood:

1. Add a `UISegmentedControl` to the `EntryDetailViewController` scene.
2. Make 3 segments.
3. Set each segment's title to a mood. In the example screen recording, it uses happy, sad, and neutral emoji for the three moods, but you can choose anything you want.
4. Make an outlet from the segmented control to the view controller's class file.

## Data Model Updates and Migration

Now, you will update your Core Data model to include a property to hold the mood that the user selects on the segmented control.

1. Select your Core Data data model file. In the menu, select Editor -> Add Model Version (keep it the same name and click the "Finish" button).
2. In the new data model file, add a new attribute called `mood`. Set its type to be `String`.
3. Give the `mood` a default value. In the example, the default value is 😐, the neutral emoji. Again, you can choose whichever 3 moods you want. Just make sure to set the default value of this attribute to one of them. This will allow the `Entry` objects that have been created before `mood` was added to have an initial value.
4. In the File Inspector with the data model file selected, set the current model version to the new model version you just created. (It should be something like `Journal 2`)
5. Now navigate to your "Entry+Convenience.swift" file where you have the `Entry` extension and convenience initializer. Update the initializer to include and property initialize the new `mood` property.
6. Update your "Create" `EntryController` method to call the updated `Entry` initializer
7. Update your "Update" `EntryController` method to include a `mood` `String` parameter so the entry's `mood` can also be updated.

You will need to update the `EntryDetailViewController` in order for the mood to get saved (or updated) to an entry.

1. In the `EntryDetailViewController` 's save entry action (where you call the "Create" and "Update" CRUD methods), check which segment is selected and create a string constant that holds the corresponding mood.
  - **NOTE:** There are a few ways to go about this. You can use the `selectedSegmentIndex` property of the segmented control to get the currently selected segment. From there, you can either create a conditional statement that will set the constant's value based on the `selectedSegmentIndex`. You could also use the `titleForSegment(at: Int)` method if the text in each segment is exactly what your moods are going to be.
  - Remember that you're dealing with strings here. Now would be a perfect time to create an enum with a case for each of your three moods. You can add string raw values to each case that holds the mood string. This will help you make sure that you're using the same three strings anywhere in the application.
2. Still in the save entry action, you will need to update the "Create" and "Update" method calls to include the mood string you just made.
3. In the `updateViews()` method, set the segmented control's `selectedSegmentIndex` based on the entry's `mood` property. Doing this programmatically will cause the segmented control have the entry's corresponding mood segment selected to the user.

At this point, take the time to test your project. Make sure that:

- Entries that you have saved before adding the `mood` property have a default `mood` value added to them.
- The segmented control in the detail view controller selects the correct mood of an entry that you view when segueing to it.
- Moods get saved correctly to entries, both newly created and updated.

## Part 2 - NSFetchedResultsController Implementation

You will now implement an `NSFetchedResultsController` to manage displaying entries on and handling interactions with the table view.

### EntryController

1. Delete or comment out the `loadFromPersistentStore` method, and the `entries` array in the `EntryController`. The

fetch results controller will manage performing fetch requests and giving data to the table view now.

### EntriesTableViewController

1. In the `EntriesTableViewController`, create a lazy stored property called `fetchResultsController`. Its type should be `NSFetchedResultsController<Entry>`. It should be initialized using a closure. Inside the closure:
  - Create a fetch request from the `Entry` object.
  - Create a sort descriptor that will sort the entries based on their `timestamp`. This can be either ascending or descending depending on your preference.
  - Give the sort descriptor to the fetch request's `sortDescriptors` property. Note that this property's type is an array of sort descriptors, not a single one.
  - Create a constant that references your core data stack's `mainContext`.
  - Create a constant and initialize an `NSFetchedResultsController` using the fetch request and managed object context. For the `sectionNameKeyPath`, put "mood" (exactly how you spelled it in the data model file), and `nil` for `cacheName`.
  - Set this view controller class as the delegate of the fetched results controller. **NOTE:** Xcode will give you an error, but you will fix it in just a second.
  - Perform the fetch request using the fetched results controller
  - Return the fetched results controller.
2. Adopt the `NSFetchedResultsControllerDelegate` protocol in this view controller.
3. Add the following delegate methods to the table view controller:
  - `controllerWillChangeContent`.
  - `controllerDidChangeContent`.
  - `didChange sectionInfo ... atSectionIndex`.
  - `didChange anObject ... at indexPath`.

Remember that the implementation of these methods is going to be the same in most cases. You can use the implementations created in this morning's guided project.

Now you will change the `UITableViewDataSource` methods to look to the fetched results controller for information about how to set up the table view instead of the (no longer existing) `entries` array in the `EntryController`.

4. Add the `numberOfSections(in tableView: ...)` method if you don't have it already. This should use the number of sections in the fetched results controller's `sections` array.
5. In the `numberOfRowsInSection`, Again, use the `section` parameter to get the section currently being set up to return the `numberOfObjects`.
6. In the `cellForRowAt`, use the fetched results controller's `object(at: IndexPath)` method to get the correct entry corresponding to the cell instead of using the `entries` array in the `EntryController`.
7. In the `commit editingStyle`, use the `object(at: IndexPath)` method again to get the correct entry to be deleted instead of using the `entries` array in the `EntryController`.
8. Use the same `object(at: IndexPath)` method in the `prepare(for segue: ...)` method to get the correct entry instead of using the `entries` array in the `EntryController`.

## Go Further

Just like yesterday, try to solidify today's concepts by starting over and rewriting the project from where you started today. Or even better, try to write the entire project with both today and yesterday's content from scratch. Use these instructions as sparingly as possible to help you practice recall.

