IvanCaldwell / **ios-journal-coredata**
forked from LambdaSchool/ios-journal-coredata

No description, website, or topics provided.

Edit

Manage topics

1 commit | 4 branches | 0 releases | 1 contributor

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

This branch is even with LambdaSchool:master. Pull request Compare

SpencerCurtis Create README.md with project instructions Latest commit `059e8a1` on Aug 12, 2018

README.md Create README.md with project instructions 5 months ago

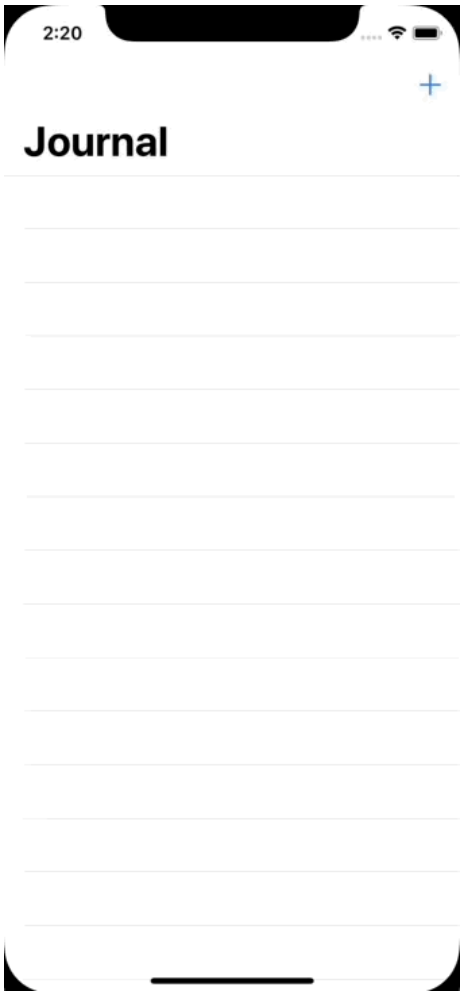
README.md

Journal (Core Data)

Introduction

This version of Journal will allow you to implement each part of CRUD when working with Core Data. This project is essentially the same as the Journal project you've previously wrote. However, please write this project from scratch. We can't regulate this, but you will undoubtedly learn less if you are copying and pasting past code.

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



Instructions

Please fork and clone this repository. This repository does not have a starter project, so create one inside of the cloned repository folder.

Part 1 - Storyboard Layout

This application will implement the Master-Detail pattern that you're surely familiar with by now.

EntriesTableViewController and EntryTableViewCell

1. Delete the view controller scene that comes with the Main.storyboard
2. Add a `UITableViewController` scene, then embed it in a navigation controller. Set the navigation controller as the initial view controller.
3. Add a `UIViewController` scene as well. Leave it blank for now.
4. On the table view controller scene, change its navigation item's title to "Journal".
5. Add a bar button item on the right side of the navigation bar. Change its "System Item" to "Add".
6. Create a segue from the bar button item to the blank `UIViewController` scene. This segue will be used to create new entries. Give it an appropriate identifier.
7. Create a second segue from the table view's prototype cell. This segue will be used to view existing entries. Give it an appropriate identifier as well.
8. Create a Cocoa Touch subclass of `UITableViewController` called `EntriesTableViewController`. Set this table view controller scene's class to it.
9. This prototype cell will be a custom cell. Add three labels. One for the entry's title, timestamp, and body text.

10. Create a Cocoa Touch subclass of `UITableViewCell` called `EntryTableViewCell`. Set this cell's class to it.
11. Create outlets for the three labels in the `EntryTableViewCell` class.

EntryDetailViewController

1. In the `UIViewController` scene, add a `UITextField`. Set its placeholder text to "Enter a title:"
2. Add a `UITextView`. Remove the Lorem Ipsum text from it. Constrain the text field right below the navigation bar, and the text view below that.
3. Add a navigation item to the view controller, then add a bar button item on the right side of the navigation bar. Change its "System Item" to "Save".
4. Create a Cocoa Touch subclass of `UIViewController` called `EntryDetailViewController`. Set this scene's class to the newly created subclass in the Identity Inspector.
5. Create an outlet from the text field and one from the text view. Also, create an action from the bar button item.

Part 2 - Entry and EntryController Setup

CoreDataStack

Create a swift file for your core data stack. Feel free to take the core data stack you used in this morning's project and paste it in this file. You may need to change the name of the persistent container to match the name of your data model file.

Entry

You will be using a model object called `Entry`.

1. Create a new Data Model file under the Core Data section. Make the name of the file match the name of your project.
2. Create a new entity and call it `Entry`. Keep the codegen as "Class Definition".
3. Add the following attributes to the `Entry` entity:
    - A `title` string.
    - A `bodyText` string.
    - A `timestamp` `Date`.
    - An `identifier` string.
4. Create a new swift file called "Entry+Convenience.swift".
5. Import `CoreData` in this file.
6. Add an extension on `Entry`.
7. Create a convenience initializer that takes in values for each of the `Entry` entity's attributes, and an instance of `NSManagedObjectContext`. Consider giving default values to the timestamp and identifier parameters in this initializer. This initializer should:
    - Call the `Entry` class' initializer that takes in an `NSManagedObjectContext`
    - Set the value of attributes you defined in the data model using the parameters of the initializer.

EntryController

1. Create a Swift file called "EntryController.swift". Make a class called `EntryController`.
2. Create a function called `saveToPersistentStore()`. This method should save your core data stack's `mainContext`. Remember that this will bundle the changes in the context, pass them to the persistent store coordinator who will then put those changes in the persistent store.
3. Create a function called `loadFromPersistentStore() -> [Entry]`. This method should:
    - Create an `NSFetchRequest` for `Entry` objects
    - Perform that fetch request on the core data stack's `mainContext` using a do-try-catch block.

    - Return the results of the fetch request.
    - In the catch statement, handle any errors and return an empty array.
4. Create an `entries: [Entry]` computed property. Inside of the computed property, call `loadFromPersistentStore()`. This will allow any changes to the persistent store become immediately visible to the user when accessing this array (i.e. in the table view showing a list of entries).
5. Create a "Create" CRUD method that will:
    - Initialize an `Entry` object
    - Save it to the persistent store.
    - **NOTE:** if Xcode is giving you a warning that the `Entry` object isn't being used, you can make the constant's name `_`, or add the `@discarableResult` attribute to the `Entry`'s convenience intializer in the extension you created.
6. Create an "Update" CRUD method. The method should:
    - Have title and bodyText parameters as well as the `Entry` you want to update.
    - Change the title and bodyText of the `Entry` to the new values passed in as parameters to the function.
    - Update the entry's timestamp to the current time as well.
    - Save these changes to the persistent store.
7. Create a "Delete" CRUD method. This method should:
    - Take an an `Entry` object to delete
    - Delete the `Entry` from the core data stack's `mainContext`
    - Save this deletion to the persistent store.

Part 3 - View and View Controller Implementation

In the `EntryTableViewCell` class:

1. Add an `entry: Entry?` variable.
2. Create an `updateViews()` function that takes the values from the `entry` variable and places them in the outlets.
3. Add a `didSet` property observer to the `entry` variable. Call `updateViews()` in it.

In the `EntryDetailViewController`:

1. Add an `entry: Entry?` variable.
2. Add an `entryController: EntryController?` variable.

In the `EntryTableViewController`:

1. Add an `entryController` constant whose value is a new instance of `EntryController`.
2. Implement the `numberOfRows` method. It should return the amount of entries in the `entryController`.
3. Implement the `cellForRowAt` method. Remember to cast the call as `EntryTableViewCell`, then pass an `Entry` to the cell's `entry` property in order for it to call the `updateViews()` method to fill in the information for the cell's labels.
4. Add the `viewWillAppear` method. It should reload the table view.
5. Implement the `commit editingStyle UITableViewDataSource` method to allow the user to swipe to delete entries. You don't have to handle the `editingStyle` being `.insert`, just `.delete`.
6. Implement the `prepare(for segue: ...)` method. If the segue's identifier shows that the user is trying to create an entry, you will only need to pass the `entryController` to the destination view controller. If the identifier shows that they want to view an entry (by tapping a cell), pass the `entryController` and also the `Entry` that corresponds with the cell they tapped.

Back in the `EntryDetailViewController`:

1. Add an `updateViews()` method. Inside of it:
    - Make sure the view is loaded.
    - Set the view controller's title to the title of the `entry` if one was passed to this view controller, or "Create Entry" if not.
    - This method should also fill in the text field and text view's `text` to the `title` and `bodyText` of the `entry` respectively.

2. Add a `didSet` to the `entry` variable, and call `updateViews()` in it. Also call `updateViews()` in the `viewDidLoad`.
3. In the bar button item's action:
    - Unwrap the text from both the text field and text view.
    - Unwrap the `entry` property separately. If there is an entry, call the `update` method in the `entryController`. If not, call the `createEntry` method in the `entryController` instead. Either way, pop the view controller off the navigation stack.

Go Further

This project will be added on to as the Sprint progresses. As such, there are no "Go Further" challenges. However it is always a good idea to rebuild this project again. The more you build a project, the more you will learn from it. If you want to challenge yourself, try to write as much as you can without referencing these instructions.