Showing and Synchronizing Data with NSFetchedResultsController

Andrew Bancroft

@andrewcbancroft www.andrewcbancroft.com

Overview

How to keep UI up-to-date with the persistent store

- Showing data in the UI
- Keeping data in sync with persistent store

Show and sync data with NSFetchedResultsController

Sync data with Notification Center and NSManagedObjectContextDidSave notification

Finish major features of ShoutOut app

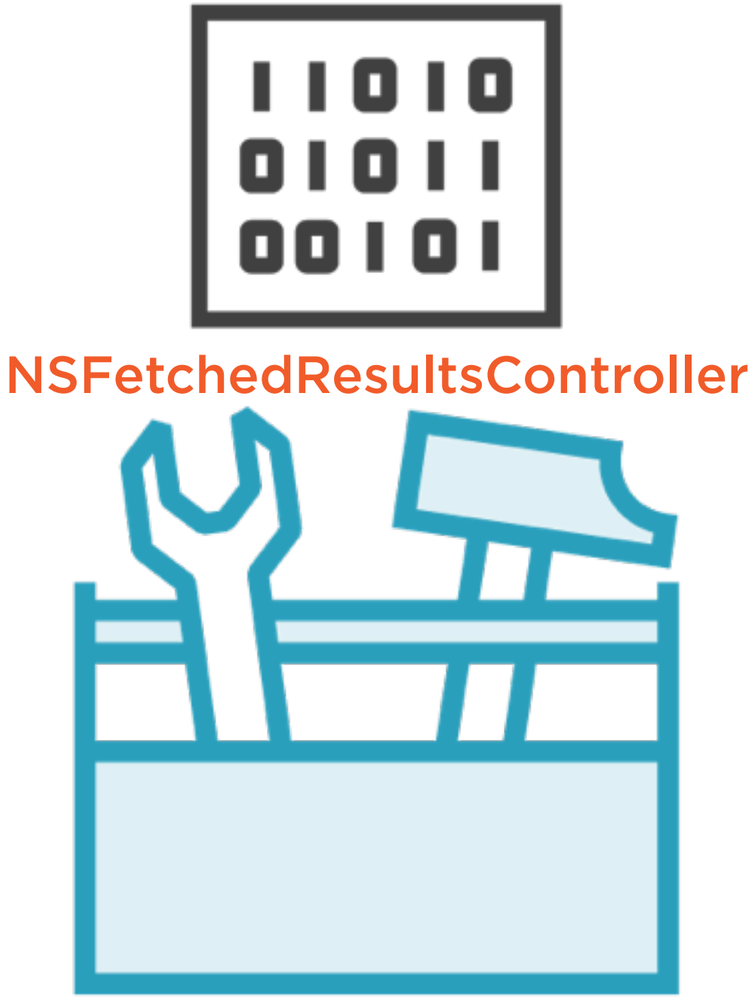
Demo

Adjust 3 areas of code

Ensure upcoming demos go smoothly

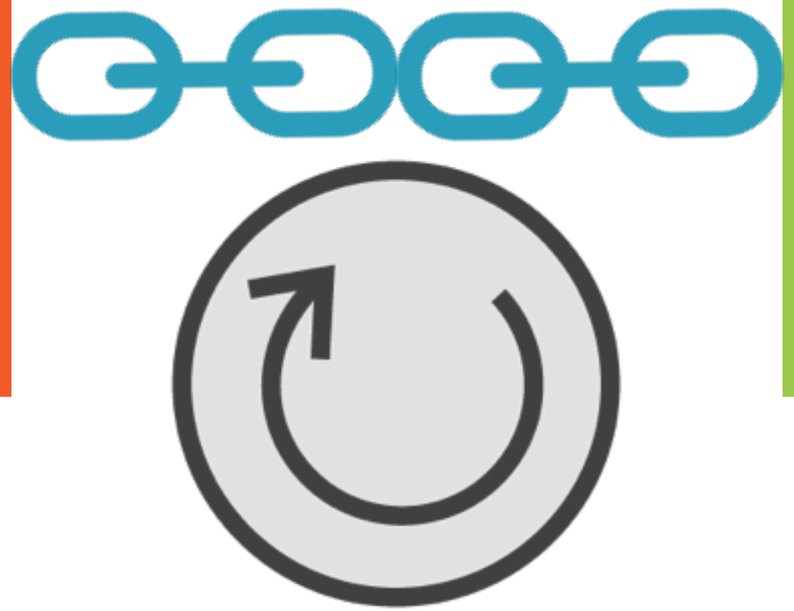
Showing Data with NSFetchedResultsController

Showing Data with NSFetchedResultsController



Why NSFetchedResultsController?

Table View

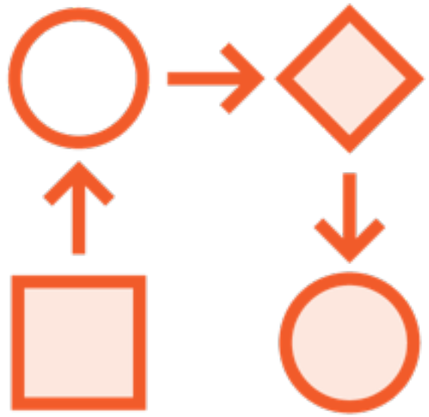


Synchronize Data

NSFetchedResultsController

NSFetchRequest

Showing Data with NSFetchedResultsController



1. First show data
2. *Then* synchronize data

```
class ShoutOutDraftsViewController:
                UIViewController,
                UITableViewDataSource,
                UITableViewDelegate {

  var fetchedResultsController: NSFetchedResultsController<ShoutOut>!


}
```

NSFetchedResultsController Logistics

Used within a view controller that uses a UITableView

View controller holds a reference to NSFetchedResultsController instance

NSFetchedResultsController uses generic typing

```
func configureFetchedResultsController {

   // Create shoutOutFetchRequest with predicate and/or sort descriptors
   self.fetchedResultsController = NSFetchedResultsController<ShoutOut>(
          fetchRequest: shoutOutFetchRequest,
          managedObjectContext: self.managedObjectContext,
          sectionNameKeyPath: nil,
          cacheName: nil)

}
```

Configure NSFetchedResultsController

Create an NSFetchRequest instance to be used with the NSFetchedResultsController

Configure any NSPredicates and NSSortDescriptors that are necessary

Initialize NSFetchedResultsController and assign to view controller's fetchedResultsController property

```
override func viewDidLoad() {

    configureFetchedResultsController()

    do {
        try self.fetchedResultsController.performFetch()
    } catch _ {}

}

// Implement UITableViewDataSource methods
```

Use NSFetchedResultsController

Configure NSFetchedResultsController in viewDidLoad()

Call fetchedResultsController.performFetch()

Implement UITableViewDataSource methods

Demo

Implement first screen of app

Display a list of all ShoutOuts in the persistent store

Demo

Practice setting up NSFetchedResultsController

Use as data source for UITableView

Initialize & configure NSFetchedResultsController

Implement UITableViewDataSource protocol methods

Demo

Tie up loose ends on details screen and in editor

Prepare for synchronizing insert, update, and delete scenarios

Implement details screen

Implement ability to edit existing ShoutOuts

Demo

Implement details screen

Synchronizing Data with NSFetchedResultsController

Synchronization Scenarios

Synchronization Scenarios

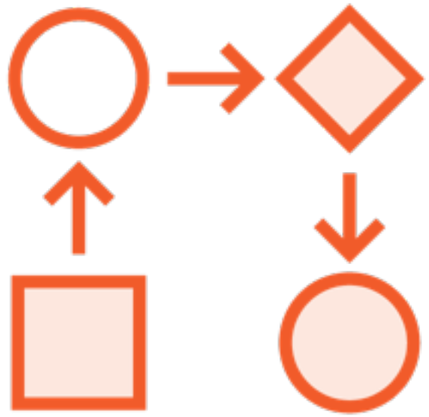
NSFetchedResultsControllerDelegate

Insert

Update

Delete

Workflow for Synchronizing Data



1. Adopt NSFetchedResultsControllerDelegate
2. Implement delegate method for responding to when the fetched results controller will change content
3. Implement delegate method for updating the UI with the changes detected by the fetched results controller
4. Implement delegate method for responding to when the fetched results controller finished changing content

Workflow for Synchronizing Data

Content *will* change

Update UI

Content *did* change

```
class ShoutOutDraftsViewController:
                UIViewController,
                UITableViewDataSource,
                UITableViewDelegate {




}
```

NSFetchedResultsControllerDelegate

Adopt the NSFetchedResultsControllerDelegate protocol

```
class ShoutOutDraftsViewController:
                UIViewController,
                UITableViewDataSource,
                UITableViewDelegate,
                NSFetchedResultsControllerDelegate {




}
```

NSFetchedResultsControllerDelegate

Conform to NSFetchedResultsControllerDelegate protocol

```
func configureFetchedResultsController {

    // Previous implementation

    self.fetchedResultsController.delegate = self


}
```

NSFetchedResultsControllerDelegate

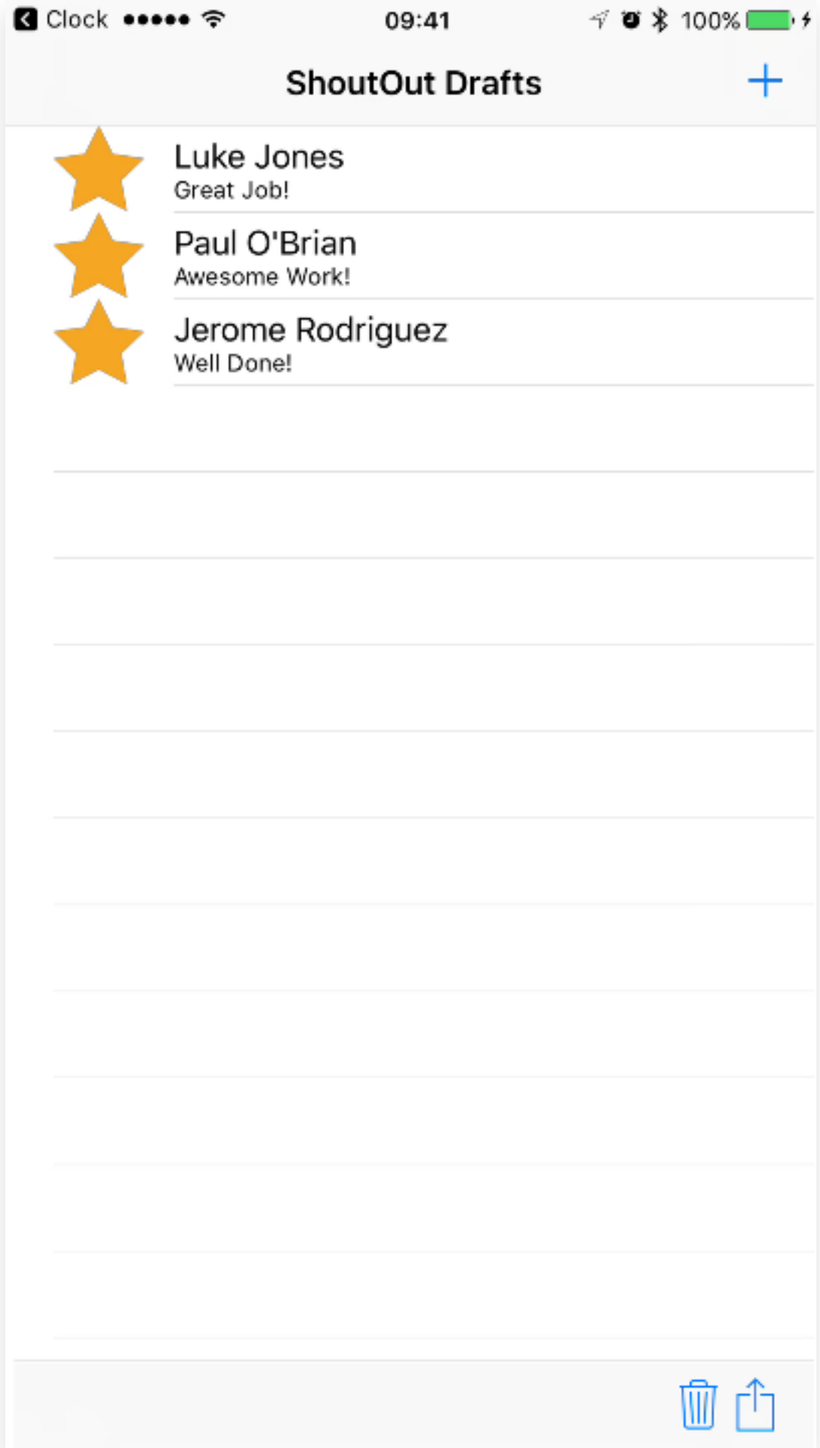
Conform to NSFetchedResultsControllerDelegate protocol

Set initialized NSFetchedResultsController instance's delegate property to self

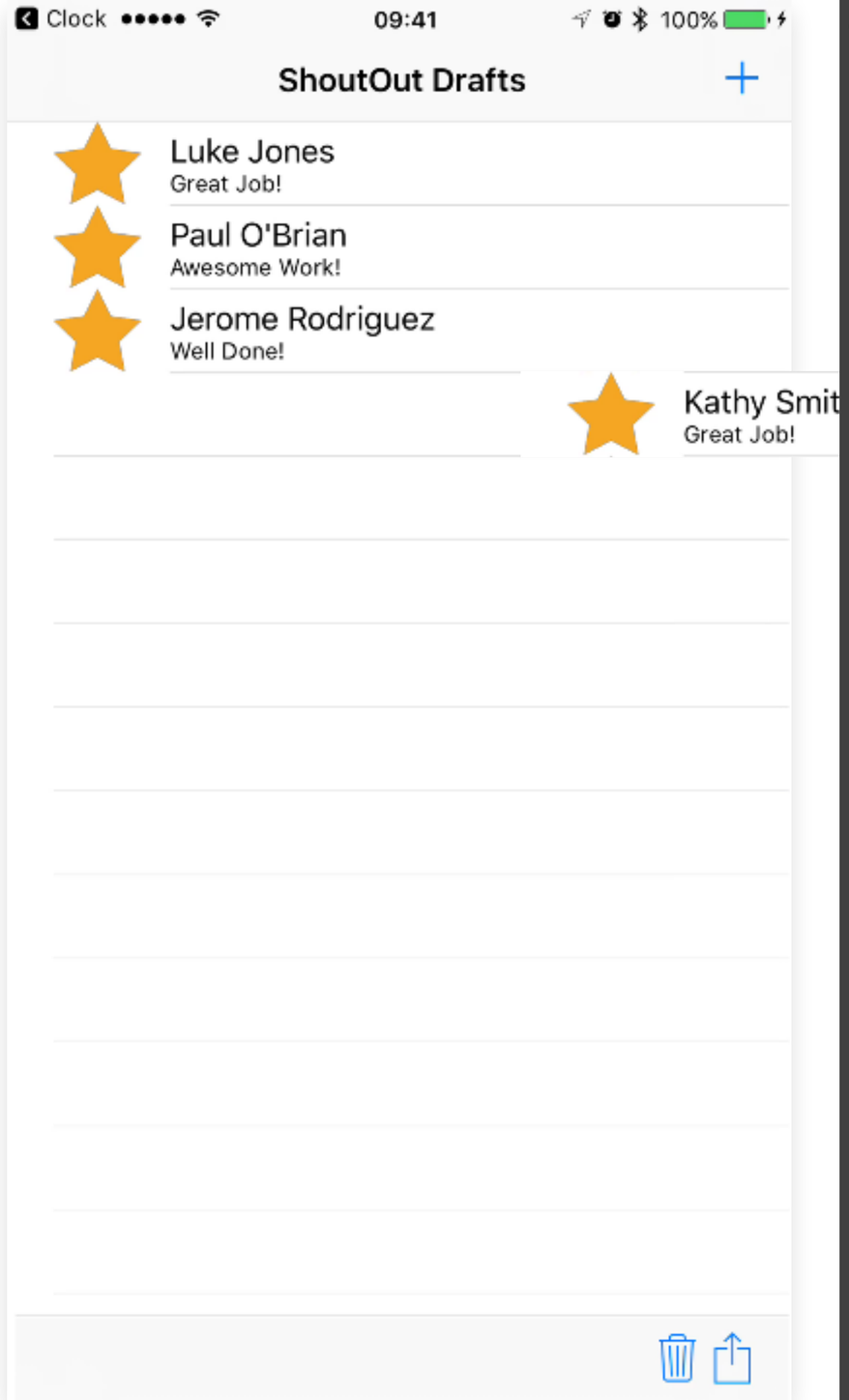
```
func controllerWillChangeContent(. . .)
{
}


func controller(
        _ controller: NSFetchedResultsController
        didChange anObject: Any,
        at indexPath: IndexPath?,
        for type: NSFetchedResultsChangeType,
        newIndexPath: IndexPath?)
{
}


func controllerDidChangeContent(. . .)
{
}
```

```
func controllerWillChangeContent(. . .)
{
  self.tableView.beginUpdates()
}

func controller(
      _ controller: NSFetchedResultsController
      didChange anObject: Any,
      at indexPath: IndexPath?,
      for type: NSFetchedResultsChangeType,
      newIndexPath: IndexPath?)
{
}

func controllerDidChangeContent(. . .)
{
  self.tableView.endUpdates()
}
```

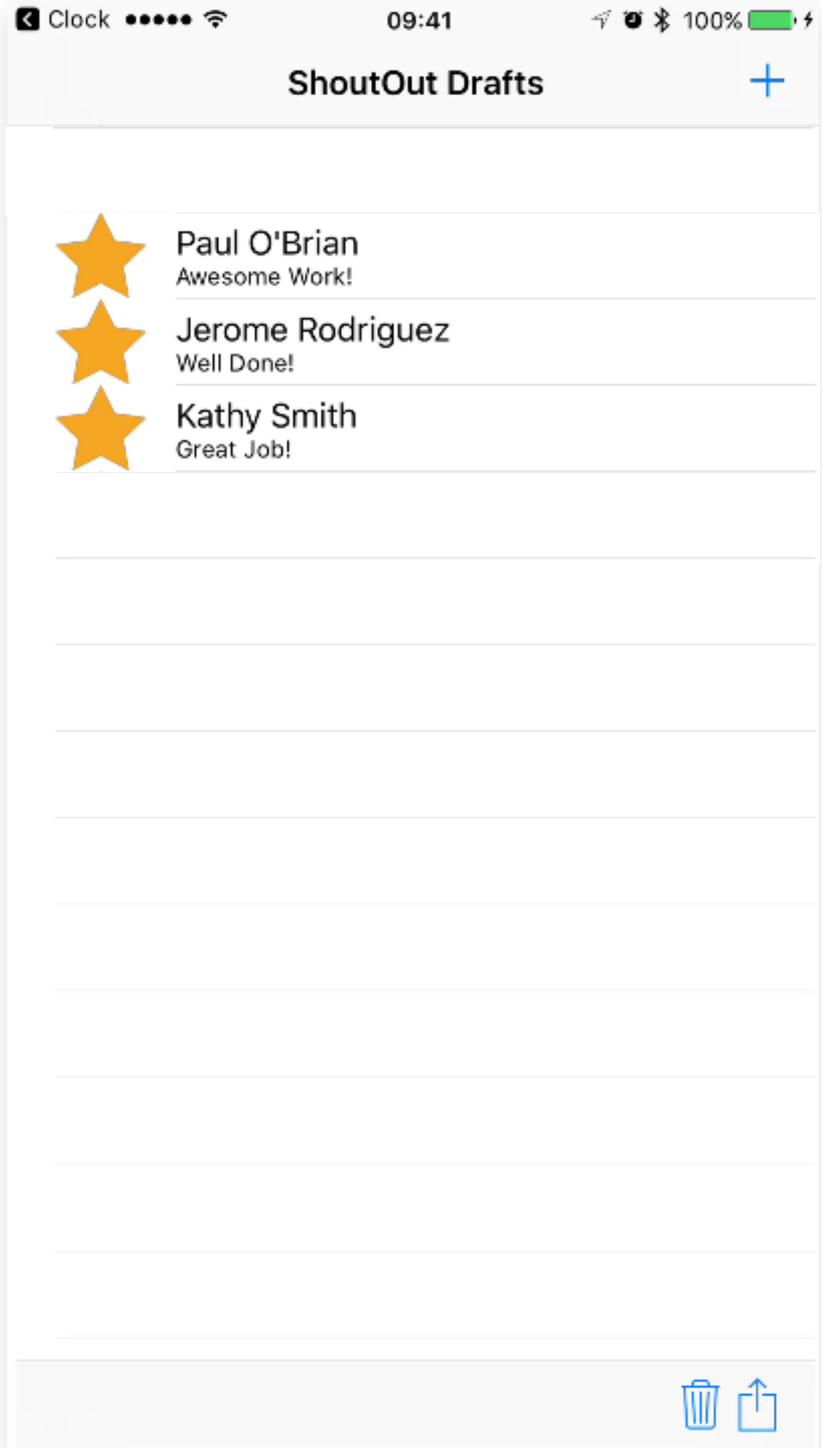
```
func controllerWillChangeContent(. . .)
{
  self.tableView.beginUpdates()
}

func controller(
       _ controller: NSFetchedResultsController
       didChange anObject: Any,
       at indexPath: IndexPath?,
       for type: NSFetchedResultsChangeType,
       newIndexPath: IndexPath?)
{
}

func controllerDidChangeContent(. . .)
{
  self.tableView.endUpdates()
}
```



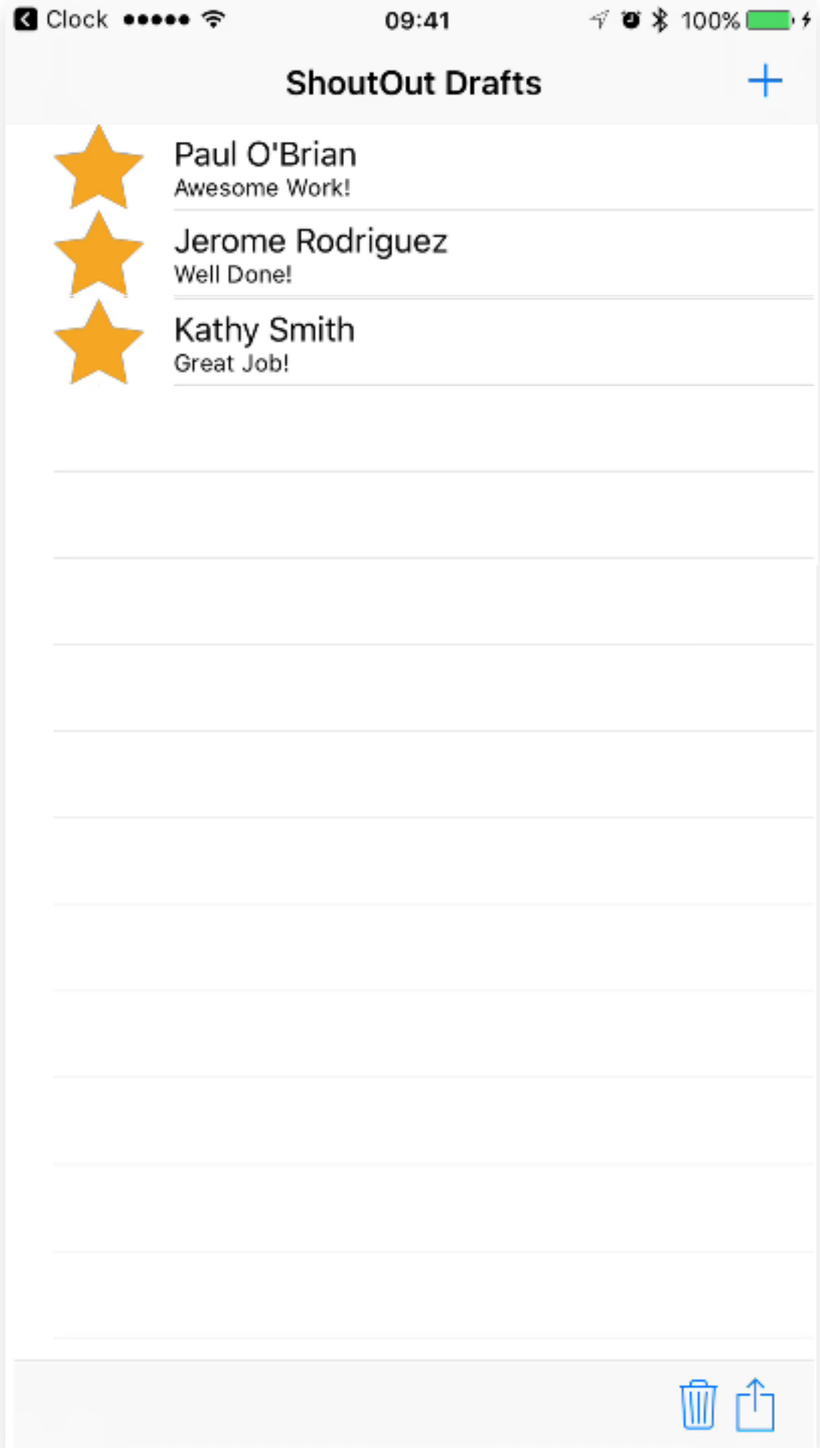

```
func controller(
        _ controller: NSFetchedResultsController
        didChange anObject: Any,
        at indexPath: IndexPath?,
        for type: NSFetchedResultsChangeType,
        newIndexPath: IndexPath?)
{
    switch type {
    case .insert:

    case .delete:

    case .update:

    case .move:

    }
}
```



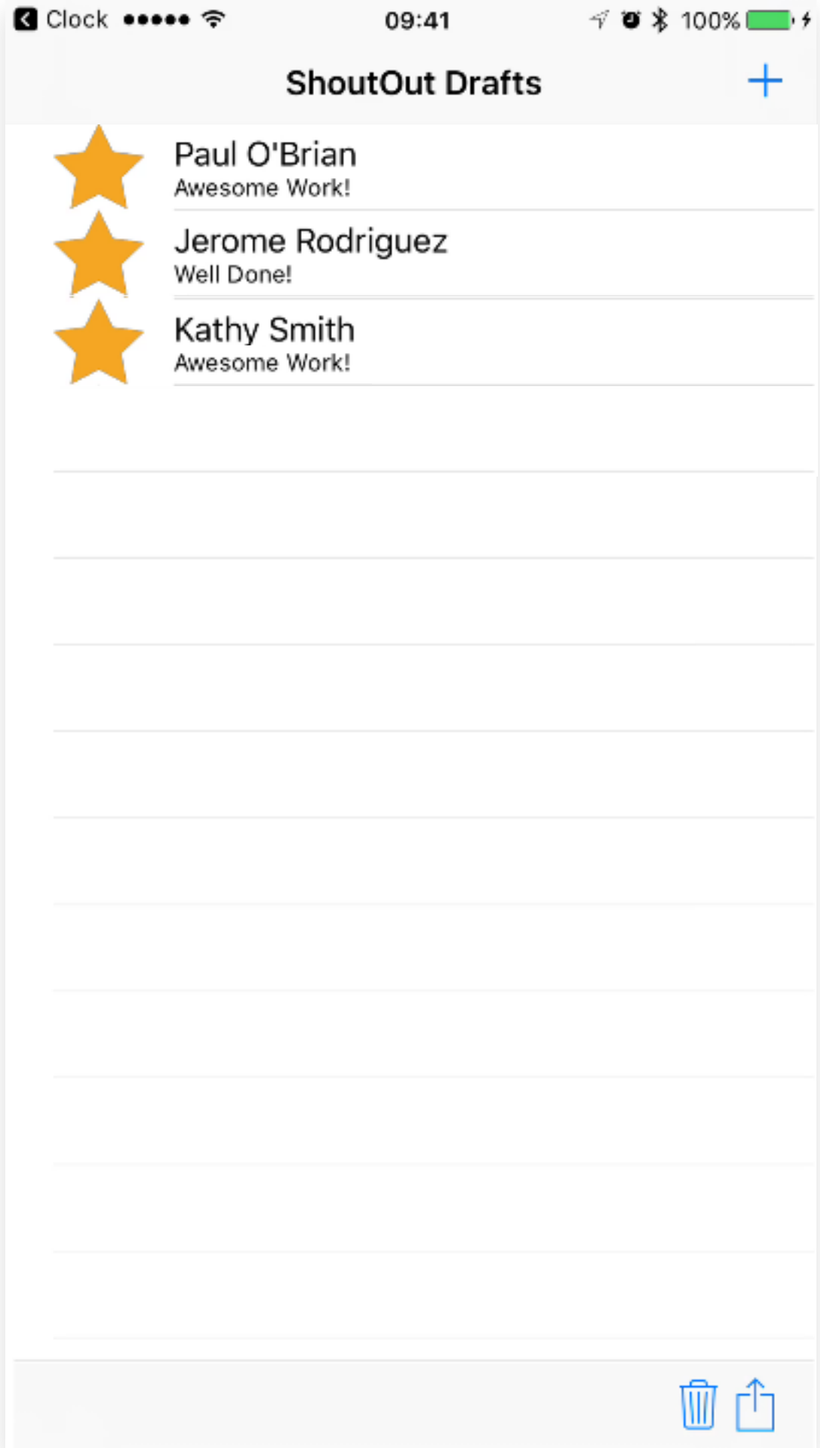

```
func controller(
        _ controller: NSFetchedResultsController
        didChange anObject: Any,
        at indexPath: IndexPath?,
        for type: NSFetchedResultsChangeType,
        newIndexPath: IndexPath?)
{
    switch type {
    case .insert:
      // Add row to table view
    case .delete:

    case .update:

    case .move:

    }
}
```



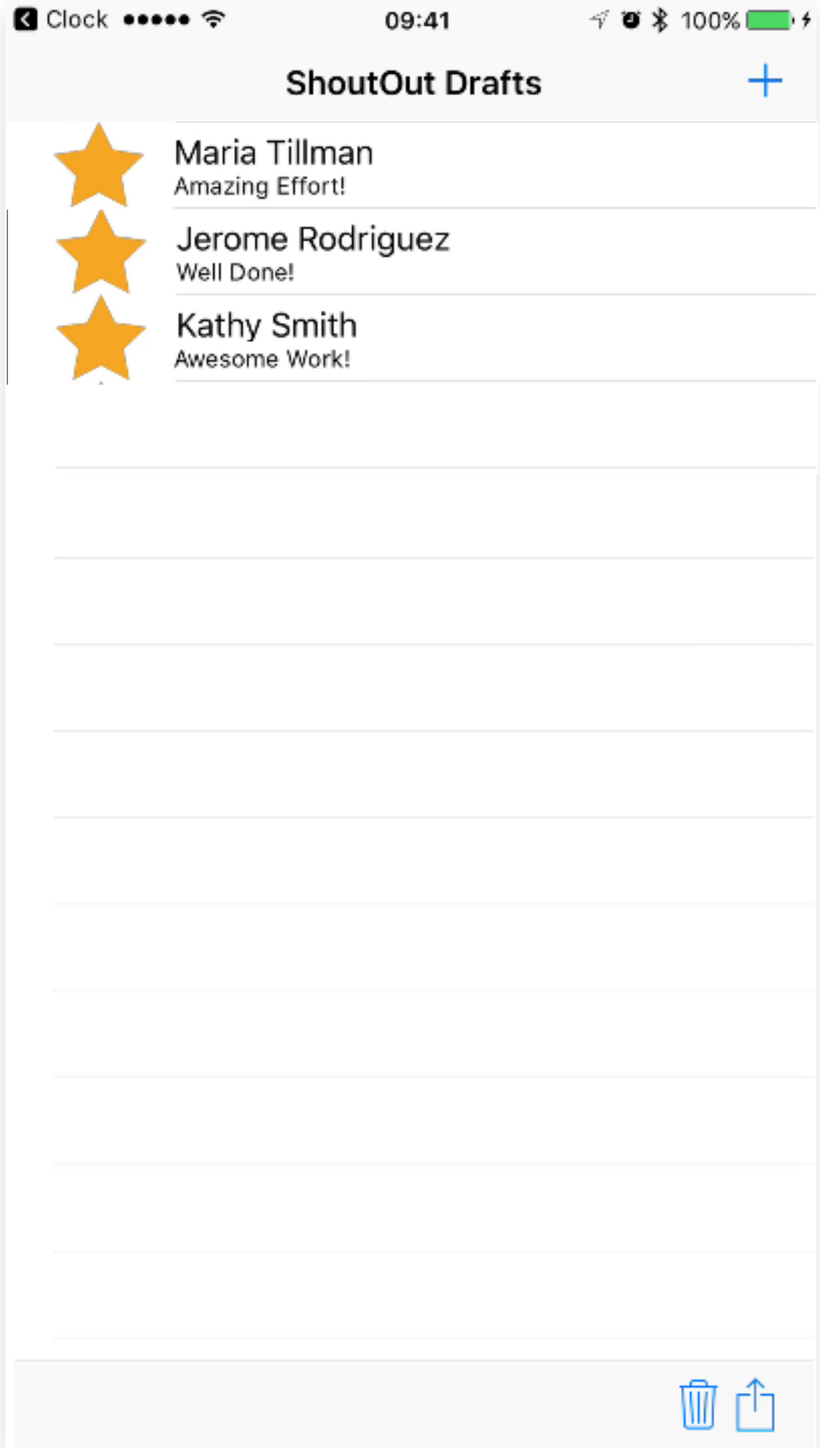

```
func controller(
        _ controller: NSFetchedResultsController
        didChange anObject: Any,
        at indexPath: IndexPath?,
        for type: NSFetchedResultsChangeType,
        newIndexPath: IndexPath?)
{
    switch type {
    case .insert:
      // Add row to table view
    case .delete:
      // Remove row from table view
    case .update:

    case .move:

    }
}
```




```
func controller(
        _ controller: NSFetchedResultsController
        didChange anObject: Any,
        at indexPath: IndexPath?,
        for type: NSFetchedResultsChangeType,
        newIndexPath: IndexPath?)
{
    switch type {
    case .insert:
      // Add row to table view
    case .delete:
      // Remove row from table view
    case .update:

    case .move:

    }
}
```




```
func controller(
        _ controller: NSFetchedResultsController
        didChange anObject: Any,
        at indexPath: IndexPath?,
        for type: NSFetchedResultsChangeType,
        newIndexPath: IndexPath?)
{
    switch type {
    case .insert:
      // Add row to table view
    case .delete:
      // Remove row from table view
    case .update:
      // Change data in table view
    case .move:

    }
}
```



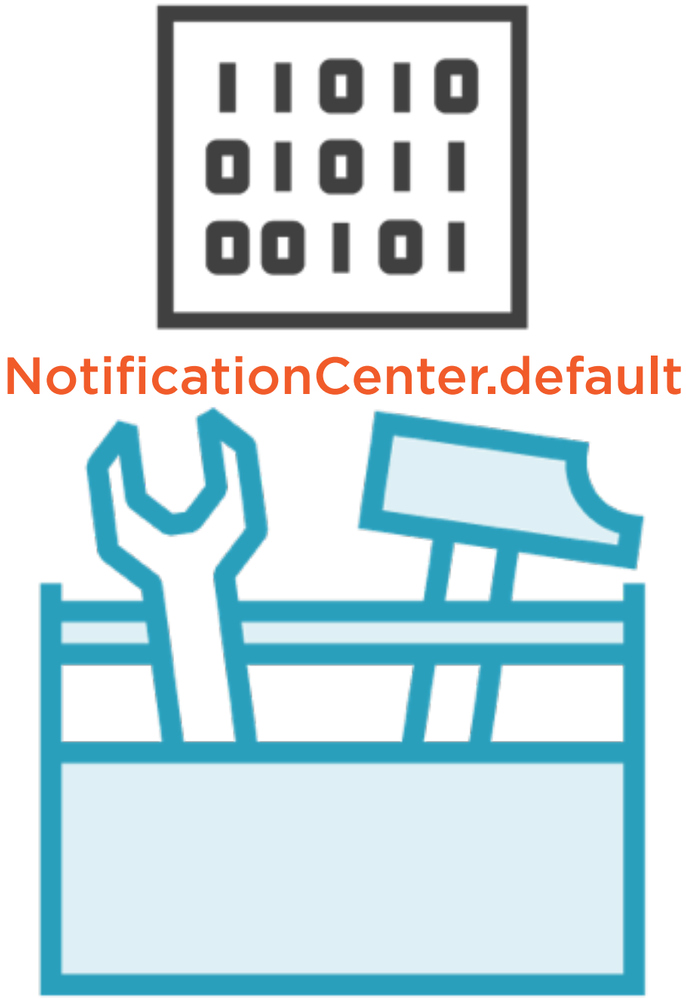

```
func controller(
        _ controller: NSFetchedResultsController
        didChange anObject: Any,
        at indexPath: IndexPath?,
        for type: NSFetchedResultsChangeType,
        newIndexPath: IndexPath?)
{
    switch type {
    case .insert:
      // Add row to table view
    case .delete:
      // Remove row from table view
    case .update:
      // Change data in table view
    case .move:
      // Rearrange rows in table view
    }
}
```

Synchronizing Data with Notification Center

How do you update a UI with a single NSManagedObject instance?

What alternative is there to NSFetchedResultsControllerDelegate?

Synchronizing Data with Notification Center



Notification

Notification

Notification

102.1 FM

94.2 FM

98.8 FM

UITableViewSelectionDidChange

NSKeyboardDidShow

UITextFieldDidChange

NSManagedObjectContextDidSave

NSManagedObjectContextDidSave

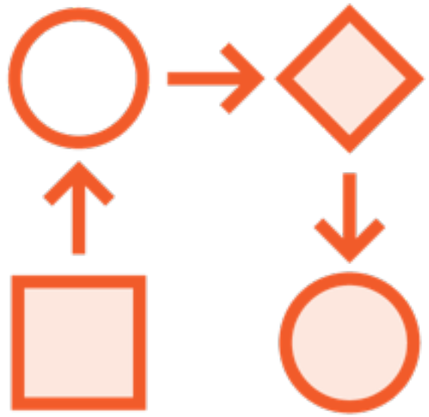
NSManagedObjectContextDidSave

NSManagedObjectContextDidSave

NSManagedObjectContextDidSave

NSManagedObjectContextDidSave

Workflow for Synchronizing Data



1. NotificationCenter.default.addObserver
2. Listen for NSManagedObjectContextDidSave
3. Implement callback closure to respond to the notification by updating the UI
4. NotificationCenter.default.removeObserver

Summary

Looked at scenarios requiring UI updates:

- Insert
- Update
- Delete

Used NSFetchedResultsController + delegate to show and sync data

Observed NSManagedObjectContextDidSave notification

Coming up: Creating new versions of your data model and implementing migrations

Have you ever made a mistake?

Have you ever had a user change his/her mind about the requirements of the app you're building?

Hard to be perfect on the first version of your app

Things outside of your control change your direction

Some changes in direction occur at the level of your data model

Summary

Looked at scenarios requiring UI updates:

- Insert
- Update
- Delete

Used NSFetchedResultsController + delegate to show and sync data

Observed NSManagedObjectContextDidSave notification

Coming up: Creating new versions of your data model and implementing migrations