Introducing

iOS 9 Search APIs

Hands-On Challenges

Introducing iOS 9 Search APIs Hands-On Challenges

Challenge 3: Handling Search Result Selection

Throughout this series you've been concentrating on how to add your own results to the Spotlight search results, whether it be through user activity indexing, or through Core Spotlight, but this is only half of the story.

In this tutorial you discovered how iOS handles when the user taps on a search result – via the app delegate. Your challenge is to extend the technique you learned to handle the other types of search results present in GreenGrocer – namely products and the store details.

Hints

- All search result interactions arrive at your app through the same app delegate method.
- You can use the existing user activity dispatch system to handle passing the new user activities to the correct view controller.
- Remember that you added the product ID to the `userInfo` dictionary so that you can work out which product you should be displaying.

Solution

Before explaining the solution in great detail, it's worth first understanding a little more of the background surrounding the activity dispatching that was introduced during the tutorial.

As you know, when a user taps a search result, the system brings your app to the foreground, and calls the `application(_:, continueUserActivity:, restorationHandler:)` on your app delegate. This is your opportunity to get the view controller hierarchy and app content to the correct state to display the search result.

To assist with this, GreenGrocer contains a couple of protocols, which allow a parent view controller to establish whether any of its children can handle a given user activity type:

```
protocol RestorableActivity {
  var restorableActivities : Set<String> { get }
}

protocol RestorableActivityContainer : RestorableActivity {
  func primaryRestorableResponderForActivityType(activityType:
String) -> UIResponder?
}
```

You can ask a `RestorableActivity` which activity names it's capable of restoring, and for container view controllers (such as tab controllers and navigation controllers) you can ask which of its children it nominates to handle a given activity type.

The project includes default implementations of these protocol methods for the container type – which involve collating the information for its children. You can check out the simple implementations in **RestorableActivity.swift**.

Since `UIResponder` defines a method which is used to handle resuming activites, once you've established which view controller should be used to restore a given activity, it's simply a matter of calling the required method.

The result of this earlier work is that adding the search handling functionality for the two additional activity types is really easy:

1. State which view controller can handle the activity.

2. Override the `UIResponder` method to actually perform the data restoration.

OK – so to the actual problem at hand. You're going to start off with the Store view controller, and move on to the individual products later.

Open StoreViewController.swift, and add the following extension at the bottom of the file:

```
extension StoreViewController : RestorableActivity {
  var restorableActivities : Set<String> {
    return Set([storeDetailsActivityID])
  }
}
```

The definition of this extension states that `StoreViewController` adopts the `RestorableActivity` protocol. This view controller is represents the `storeDetailsActivityID` user activity, so this is what you return in the `restorableActivities` property set.

For the store view controller, that's all you need to do! The activity dispatch system will find locate the view controller using the activity ID, and then show it. Since this activity just represents this view controller, no further info is needed.

Build and run to test this. Use Spotlight to search for "**emporium**", and tap on the "**Ray's Fruit Emporium**" result. This will now take you right to the store page in the GreenGrocer app – sweet!

Next up, move your attention to the products – open **ProductTableViewController.swift** and add the following extension to the end of the file:

```
extension ProductTableViewController : RestorableActivity {
  var restorableActivities : Set<String> {
    return Set([productActivityName])
  }
}
```

Once again, you're using the `RestorableActivity` protocol to specify that this view controller will handle the product activities. However this time, since each activity represents a different product, you need to do some additional work.

Add the following utility method to the extension:

```
private func displayVCForProductWithId(id: String) {
  // 1:
  guard let id = NSUUID(UUIDString: id),
    let productIndex = dataStore?.products
                       .indexOf({ $0.id.isEqual(id) }) else {
      return
  }
  // 2:
  tableView.selectRowAtIndexPath(
    NSIndexPath(forRow: productIndex, inSection: 0),
```

```
      animated: false, scrollPosition: .Middle)
    // 3:
    performSegueWithIdentifier("DisplayProduct", sender: self)
  }
```

This defines a function that displays the product view controller for a product with a specified ID.

1. The ID is supplied as a string, so first it is converted into an `NSUUID`, and then the index of the product with that ID is found in the data store product list.
2. Select the appropriate row in the table view. This ensures that when the user hits back, the table view is in a consistent state, and that the segue will work correctly.
3. Perform the named segue to actually display the product view controller, with the appropriate product selected.

Now you've got this method created, all that is left is to use it from the appropriate `UIResponder` method. Add the following override to the extension:

```
override func restoreUserActivityState(activity: NSUserActivity) {
  switch activity.activityType {
  case productActivityName:
    if let id = activity.userInfo?["id"] as? String {
      displayVCForProductWithId(id)
    }
  default:
    break
  }

  super.restoreUserActivityState(activity)
}
```

This is the method that will get called with the user activity once it has been dispatched to this view controller.

First it checks that the activity is of the type it understands, before attempting to extract the product ID from the `userInfo` dictionary – remember you specified this when you created the user activity in the first video. If this is successful, it hands it off to the utility method to show the appropriate product.

That's all there is to it. Build and run your app to test it out. Use Spotlight to search for a product such as "**banana**", and then tap the result once it's shown. This will take you directly to the banana product page within the app. Magic!