Beginning Realm on iOS

Hands-on Challenges

Beginning Realm on iOS Hands-On Challenges

Challenge A: Wire up the table cells

So far you show the task's title in each row of your table view but that's simply not enough when it comes to building effective To-Do apps :]

In this quick challenge you will use all the cell outlets, already defined and connected, to show more information about a task.

Open **TaskCell.swift** and a new property and a method that takes a `Task` object and configures the cell's UI using its data:

```
var taskId: String?

func configureWithTask(task: Task) {
    taskId = task.taskId

    check.selected = task.done
    title!.text = task.title
    priority!.text = task.priorityText
    priority!.textColor = task.priorityColor
    spinner.hidden = true
}
```

The method sets the cell title text and it sets the selected state of the checkbox button included with the cell. Then it moves on to using the dynamic properties `priorityText` and `priorityColor` to adjust the cell's priority label.

Finally it hides the activity indicator, which is included with the custom table cell. (You will use that activity indicator in a later video – for now just hide it and let it be.)

Note how you store the `taskId` of the `Task` object in a separate property. You can pass around `Object` instances only if you don't switch threads, e.g. your table view delegate method that creates the cell will be called on the main thread and you will in turn call `configureWithTask(_)` also on the main thread.

Since you will perform some tasks on the task from background queues later on in the series you can't just store a reference to the `Task` object itself, you need the object id, which you will use to fetch the object again from a background thread. But let's not dig too much into that right now – you will cover that in a later video.

Open **TasksViewController.swift** and scroll to `tableView(_:, cellForRowAtIndexPath:)`. Find:

```
cell.textLabel!.text = task.title
```

Replace that line with:

```
cell.configureWithTask(task)
```

When you run the app now you will see the cells show detailed information about each task you create like so:

Challenge B: Add one more Realm model class

To keep excercising, you will add one more Realm model class. Open **User.swift** and add inside:

```
import Foundation
import RealmSwift

class User: Object {
  dynamic var name = ""

  convenience init(name: String) {
    self.init()
    self.name = name
  }
}
```

This new data entity has a single `String` property called `name`. Later on you will add more to it but at this point that will suffice. You also add a convenience `init` to make creating objects of that type a bit easier.

Now that you have two data entities it'd be nice if you had some test data too. This is a key step when you develop data driven apps since you always need a minimal set of data to test the app with.

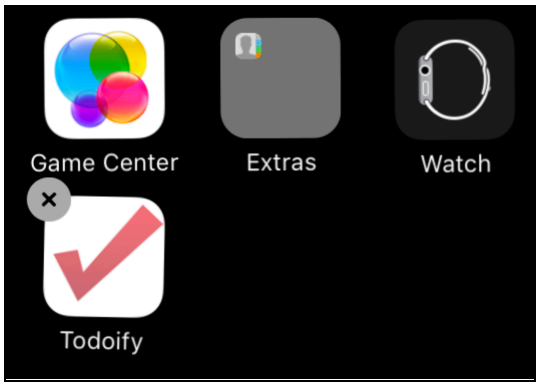
Open **TestData.swift** and uncomment all the code inside the `defaults()` method.

The code in question checks if there are any objects stored in your app's realm by using the `isEmpty` property:

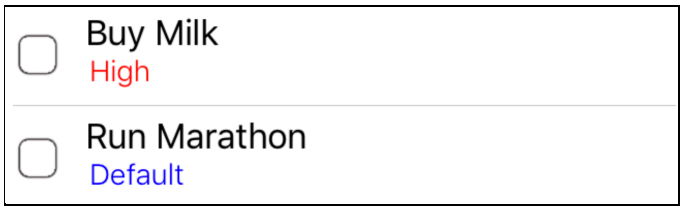
```
guard realm.isEmpty else {return}
```

If your app's realm doesn't contain any data the method adds two tasks called "Buy Milk" and "Run Marathon" and two users: "Me" and "Others".

NB: To test this code you need to click and hold on Todoify's icon in the Simulator until you see a "jiggling" animation and delete buttons appear:



Click on the X button and delete the app. Next time you run the Xode project you will see the two default tasks appear in the task list:



Now that you have two users in your realm let's add them to the Create task view controller. Open **CreateTaskViewController.swift** and add a new property:

```
let users: [User] = {
  let realm = try! Realm()
  return Array(realm.objects(User))
```

```
}()
```

`users` is an array containing the `User` objects found in your app's realm. Note how easy it is to convert a `Results` object to an `Array` – just create a new array and pass a `Results` instance to the `init`.

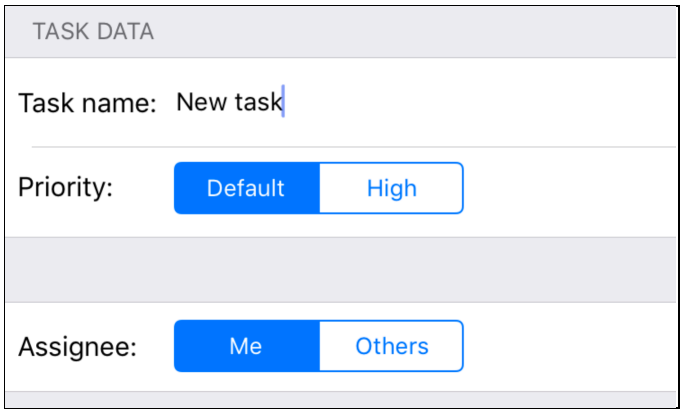
Sometimes you might want to use an Array instead of a Results object in order to preserve the order and identity of the objects in the collection. (You will learn more about Realm's dynamic collections in a later video.)

Next – scroll to viewWillAppear(_:) and add the code to populate the taskUsers segment control with the users' names:

```
taskUsers.removeAllSegments()
for user in users {
  taskUsers.insertSegmentWithTitle(user.name, atIndex:
    taskUsers.numberOfSegments, animated: false)
}
taskUsers.selectedSegmentIndex = 0
```

You remove all segments and then you loop over `users` and add a segment for each user object. Finally you select the very first segment.

Run the app and tap on the **+** button to see the Create task screen pop:



Nice! Now you can assign tasks not only to yourself but also to other people. Of course you'll have to also ask these other people to perform the task as well but hey – not everything can be achieved with an app :]