Networking with NSURLSession

Hands-on Challenges

Networking with NSURLSession Hands-On Challenges

Copyright © 2016 Razeware LLC.

All rights reserved. No part of this book or corresponding materials (such as text, images, or source code) may be reproduced or distributed by any means without prior written per- mission of the copyright owner.

This book and all corresponding materials (such as source code) are provided on an "as is" basis, without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose, and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in action of contract, tort or otherwise, arising from, out of or in connec- tion with the software or the use or other dealings in the software.

All trademarks and registered trademarks appearing in this book are the property of their respective owners.

Challenge 3: Background Sessions

Some network requests can take a long time, especially if you're downloading a large file or on a slow network. It's unreasonable to expect your user to sit there and babysit the request until it's done. Many times your users will start a long request and then background your app and expect the request to finish. In the demo, you implemented a getImage method. In this challenge, you'll implement `getImageInBackground`, which will use a background session for the request.

Read on for the solution.

A New Session

First, you're going to create a new `NSURLSession` to use for your background requests. This session will have a different configuration from the session you already have. In **NetworkClient.swift**, add the following line after the current `urlSession` property:

```
private var backgroundSession: NSURLSession
```

Now, you need to add code to init to setup this session. Add the following code to the end of init:

```
let backgroundConfiguration = NSURLSessionConfiguration
  .backgroundSessionConfigurationWithIdentifier(
    "com.razeware.flickrfeed")
backgroundSession = NSURLSession(configuration:
  backgroundConfiguration, delegate: self, delegateQueue: nil)
```

When creating a background session configuration, you must supply a unique identifier. Here, you use the bundle id of the app since you'll only have one background session. If your app is terminated by the system and relaunched, you can use the identifier to create a new session and complete the transfer(s). Note that if your app is terminated by the user, background transfers will be canceled.

One of the requirements for a background session is that it *has* to use the delegate method for handling data. It cannot use the completion handler method.

When you pass `self` to the `NSURLSession` initializer, the compiler complains that `NetworkClient` is not a `NSURLSessionDelegate`, so let's fix that. It's a good idea to add delegate code in an extension to keep it grouped together and separate from the rest of the `NetworkClient` code. At the very bottom of the file, add this code:

```
extension NetworkClient: NSURLSessionDelegate,
  NSURLSessionDownloadDelegate {
}
```

Now you'll see a couple of errors, including one that `NetworkClient` does not conform to `NSObjectProtocol`. The simplest way to fix that is to make `NetworkClient` inherit from `NSObject`. Change the declaration of `NetworkClient` from this:

```
class NetworkClient  {
```

To this:

```
class NetworkClient: NSObject {
```

That fixes the problem, but requires some modifications to the init method. Change the declaration to an override:

```
override init() {
```

And add a call to super, just after setting urlSession:

```
super.init()
```

Now you have a catch 22. Swift requires that you initialize `backgroundSession` before calling `super.init()`, but you must use `self` to initialize it and you can't use `self` before the call to `super.init()`. The fix is to use an implicitly unwrapped optional for backgroundSession. Change the declaration to this:

```
private var backgroundSession: NSURLSession!
```

Now, before you implement the method to get images in the background, you need a way to keep the completion handlers. You can safely assume that there will only be one completion handler per URL, so you can use a dictionary with the URL as the key and the closure as the value. Add the declaration right under the backgroundSession property:

```
private var completionHandlers = [NSURL: ImageResult]()
```

Then add this method after the existing getImage method:

```
func getImageInBackground(url: NSURL, completion: ImageResult?) ->
  NSURLSessionDownloadTask {
    completionHandlers[url] = completion
    let request = NSURLRequest(URL: url)
    let task = backgroundSession.downloadTaskWithRequest(request)
    task.resume()
    return task
}
```

Now, you just need to implement the delegate methods. Add the following code to the extension at the bottom of the file:

```
func URLSession(session: NSURLSession, task: NSURLSessionTask,
  didCompleteWithError error: NSError?) {
  // 1
  if let error = error, url = task.originalRequest?.URL,
    completion = completionHandlers[url] {
      // 2
      completionHandlers[url] = nil
      // 3
      NSOperationQueue.mainQueue().addOperationWithBlock {
        completion(nil, error)
      }
    }
  }
```

Taking this point-by-point:

1. In this method, you're only looking for an error. You'll handle a successful download in a different method. If there was an error, you get the completion handler that goes with the request URL
2. You don't want to keep the completion handler in the dictionary, so you clear it out here.
3. Finally, you call the completion handler, passing along the error. Most of the time, you want this code to operate on the main thread, so the network class is responsible for calling the completion block on the main thread.

Now, for the success scenario, add this method next:

```
  func URLSession(session: NSURLSession, downloadTask:
    NSURLSessionDownloadTask, didFinishDownloadingToURL location:
    NSURL) {
    // You must move the file or open it for reading before
    // this closure returns or it will be deleted
    // 1
    if let data = NSData(contentsOfURL: location),
      image = UIImage(data: data),
      request = downloadTask.originalRequest,
      response = downloadTask.response {
      // 2
      let cachedResponse = NSCachedURLResponse(response: response,
        data: data)
      self.urlSession.configuration.URLCache?
        .storeCachedResponse(cachedResponse, forRequest: request)
      // 3
      if let url = downloadTask.originalRequest?.URL,
        completion = completionHandlers[url] {
        completionHandlers[url] = nil
        NSOperationQueue.mainQueue().addOperationWithBlock {
          completion(image, nil)
        }
      }
    } else {
      // 4
      if let url = downloadTask.originalRequest?.URL,
        completion = completionHandlers[url] {
        completionHandlers[url] = nil
        NSOperationQueue.mainQueue().addOperationWithBlock {
          completion(nil, NetworkClientError.ImageData)
        }
      }
    }
  }
```

Here's what's going on:

1. The delegate method passes the URL of the file to which the data was downloaded. If the file can be read and parsed into an image, the operation was a success.
2. Download tasks don't cache their results by default. Since you're not saving the file from the download, it's helpful to manually cache the result.
3. Now you just have to get the completion handler for this request and pass the image to it.
4. If there was a problem creating an image from the downloaded data, you pass an error to the completion handler telling it what kind of error happened.

Now, in **PhotoCell.swift**, change the call from `getImage` to `getImageInBackground`. You're almost done, just one thing left. When the system is done downloading all background tasks for a session, it will notify your app delegate with a completion handler that you should call when you're done processing the downloads. In **AppDelegate.swift**, add this property declaration at the top:

```
var backgroundSessionCompletionHandler: (() -> Void)?
```

And add this implementation at the bottom:

```
func application(application: UIApplication,
  handleEventsForBackgroundURLSession identifier: String,
  completionHandler: () -> Void) {
    backgroundSessionCompletionHandler = completionHandler
}
```

That will save the completion handler to be used in a minute. Now, back in NetworkClient.swift, add this session delegate implementation at the bottom of the extension:

```
func URLSessionDidFinishEventsForBackgroundURLSession(session:
  NSURLSession) {
  if let appDelegate = UIApplication.sharedApplication()
    .delegate as? AppDelegate, completionHandler =
    appDelegate.backgroundSessionCompletionHandler {
      appDelegate.backgroundSessionCompletionHandler = nil
      completionHandler()
  }
}
```

The session delegate also gets notified when all the background events are done. In this method, you call the completion handler that was passed in to the app delegate. This allows the system to do things like update the snapshot for your app. If you're testing this out, it's best to test on a device. If you start the image download, background the app, and wait for a while, when you come back in you should notice that most or all of the image downloads are complete.