UDACITY

PROJECT REVIEW

CODE REVIEW 7

NOTES

SHARE YOUR ACCOMPLISHMENT!

Requires Changes

4 SPECIFICATIONS REQUIRE CHANGES

Hi Hoff! Excellent job! There are still some issues to work out, but your project is really good.

In response to your notes:

I didn't have any issues with the images loading for collections were the album was already downloaded, even after I closed the app and reopened it. However, I know this can be tough thing to deal with. In the iOS Persistence Course that goes with Virtual Tourist, look at the ImageCache.swift file in Favorite Actors. It is a really great guide for handling images!

For using completion handlers, I really like this tutorial from Grok Swift:

[**Grok Swift: Completion Handlers in Swift**]

That website is full of useful information for anything in Swift!

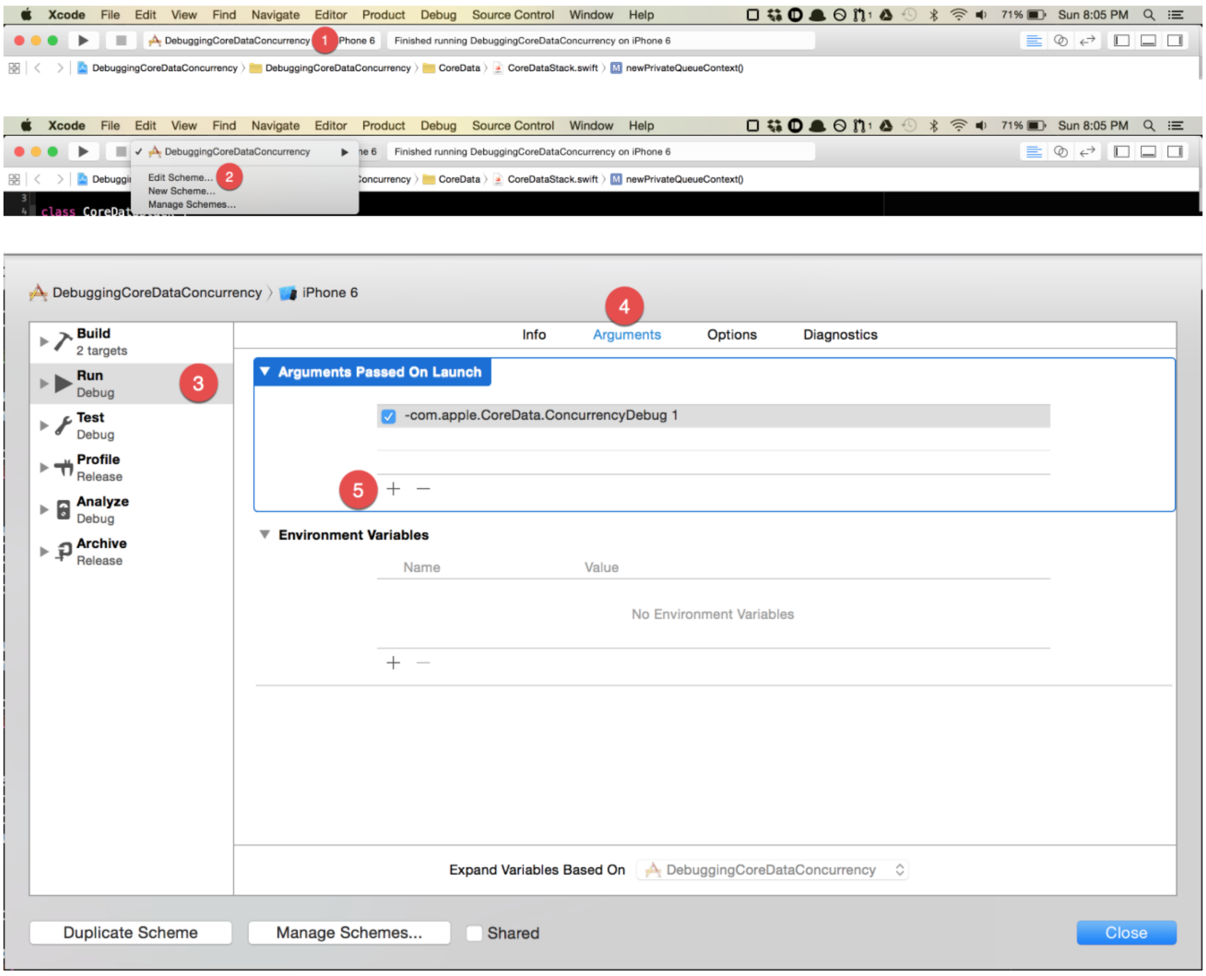
Once the problems I mentioned are addressed, you'll be ready to resubmit and should exceed specs Good luck!

Core Data

The app contains a correctly configured managed object context backed by a local SQLite store.

The CoreDataStackManager is looking great! Unfortunately, there is an area that needs just a little attention to get it functioning perfectly. Though it can be tough to spot without the right tool, it appears the context is being accessed on multiple threads, which can cause lots of problems. In the documentation, Apple warns us of this issue with concurrency while setting up Core Data. Luckily, there are special queues built right into the framework that allow us to manage this. But as they say, the first step to fixing things is realizing there's a problem. When working with Core Data, there is an extremely helpful concurrency debugger that can be enabled to make sure there are no issues.

Pawan Poudel has put together a **really great guide** for implementing both the debugger and a technique for setting up the stack to handle concurrency. This is a really important concept when dealing with Core Data, but I know you can do it!


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
DebuggingCoreDataConcurrency
1
iPhone 6
Finished running DebuggingCoreDataConcurrency on iPhone 6
DebuggingCoreDataConcurrency 〉 DebuggingCoreDataConcurrency 〉 CoreData 〉 CoreDataStack.swift 〉 newPrivateQueueContext()
Edit Scheme...
2
New Scheme...
Manage Schemes...
DebuggingCoreDataConcurrency 〉 iPhone 6
4
Info
Arguments
Options
Diagnostics
Build
2 targets
Run
Debug
3
Test
Debug
Profile
Release
Analyze
Debug
Archive
Release
Arguments Passed On Launch
5
Environment Variables
Name
Value
No Environment Variables
Expand Variables Based On
DebuggingCoreDataConcurrency
Duplicate Scheme
Manage Schemes...
Shared
Close

[✓] -com.apple.CoreData.ConcurrencyDebug 1

In case you're interested, the concurrency section of Apple's Core Data Guide can be found **here.**

✓

The app uses a managed object model created in the Xcode Model Editor. A `.xcdatamodeld` model file is present.

☆

The object model contains both Pin and Photo entities.

Both the Pin and Photo entities are present, and you've even represented map region data in your model too! Fantastic!

That said in future projects, simple data like map region values should be persisted using other techniques. NSUserDefaults is a great option for storing basic data types that are used for things just like this. NSHipster has a great post that covers different options for saving information. If you haven't seen it, it's an interesting read!

NSHipster: NSCoding/NSKeyedArchiver

✓

The object model contains a one-to-many relationship between the Pin and Photo entities, with an appropriate inverse.

Travel Locations Map View

✓

The app contain a map view that allows users to drop pins with a touch and hold gesture.

Touch and hold on the map and BAM!.. a pin drops, just like it should! Also, I really love how you customized the pin and its annotation. Outstanding! Now you should go for exceeding specs. Here’s a great post on Stack Overflow that helps explain just how this is done!

Stack Overflow: iOS Swift MapKit making an annotation draggable by user

✓

When a pin is tapped, the app transitions to the photo album associated with the pin location.

When pins are dropped on the map, the pins are persisted as Pin instances in Core Data and the context is saved.

It appears pins are only persisted to Core Data after the search for photos from Flickr is completed **and** if the search returns photos. For this spec, when a user drops a pin on the map, a pin instance should be created and the context immediately saved.

Photo Album View

☆

When a Photo Album View is opened for a pin that does not yet have any photos, it initiates a download from Flickr.

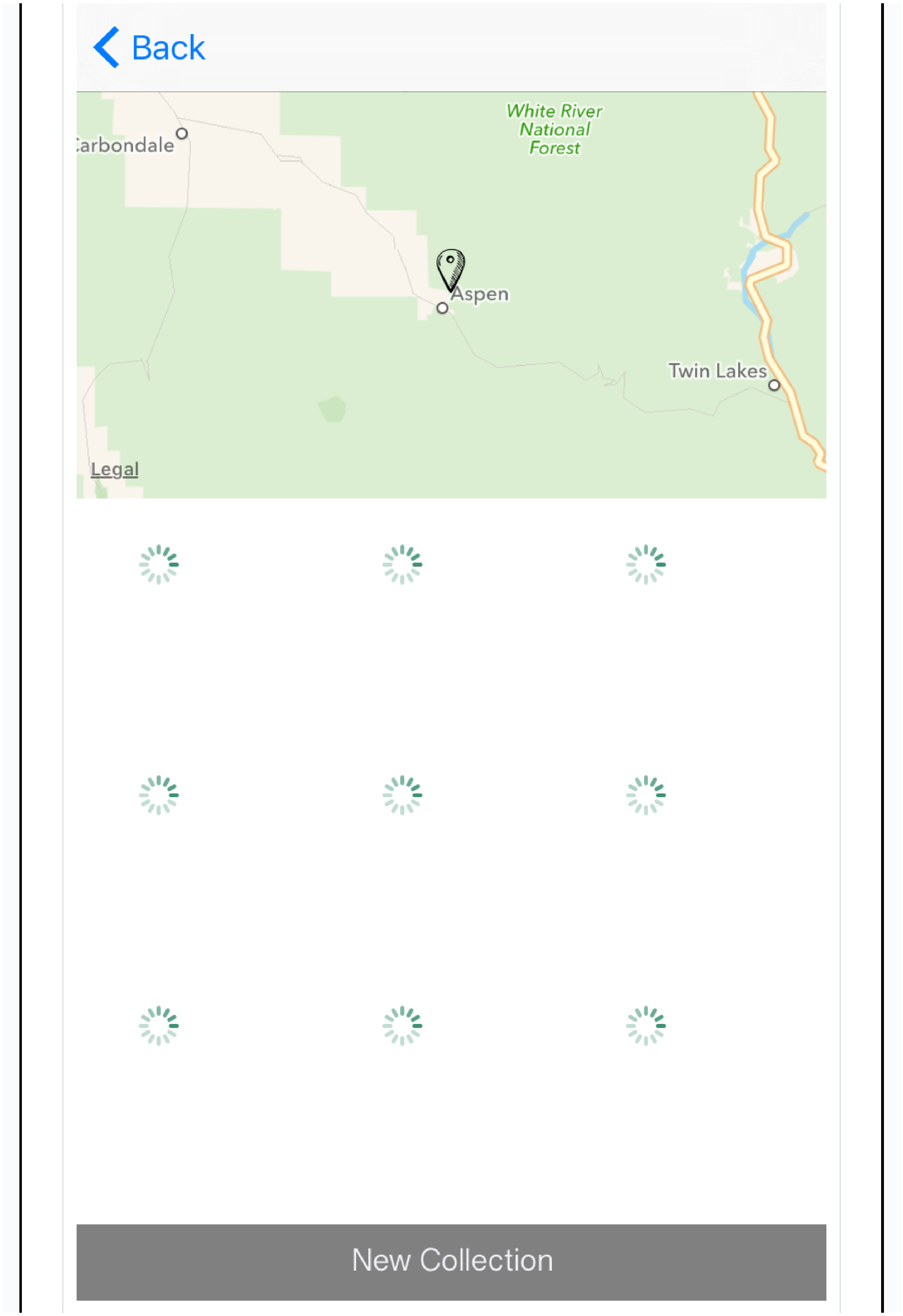
The photo search is started as soon as the user drops a pin, which is great! When you address the issue above regarding the saving a pins, still make sure to beginning fetching the photos as soon as the pin is created and saved. It really speeds up the process and that means less what time for the user!

✓

The code for downloading photos is in its own class, separate from the PhotoAlbumViewController.

Images display as they are downloaded. They are shown with placeholders in a collection view while they download, and displayed as soon as possible.

There seems to be an issue loading images. I think this is probably due to the concurrency problem mentioned above. When an album is opened the photos never seem load.


Back
White River
National
Forest
Carbondale
Aspen
Twin Lakes
Legal
New Collection

If I go back to the map, and then tap the pin again, the full loaded album appears.

✓

Image data is stored in files in the Documents directory. Corresponding paths are stored in Photo instances in Core Data, with a relationship to the pin.

This can be tricky, but you got it!

✓

Once all images have been downloaded, the user can remove photos from the album by tapping the image in the collection view. Tapping the image removes it from the photo album, the booth in the collection view, and Core Data.

I really like the extra details you put into this app. The detail view you added is really nice, and I also really enjoy the long press to delete! :)

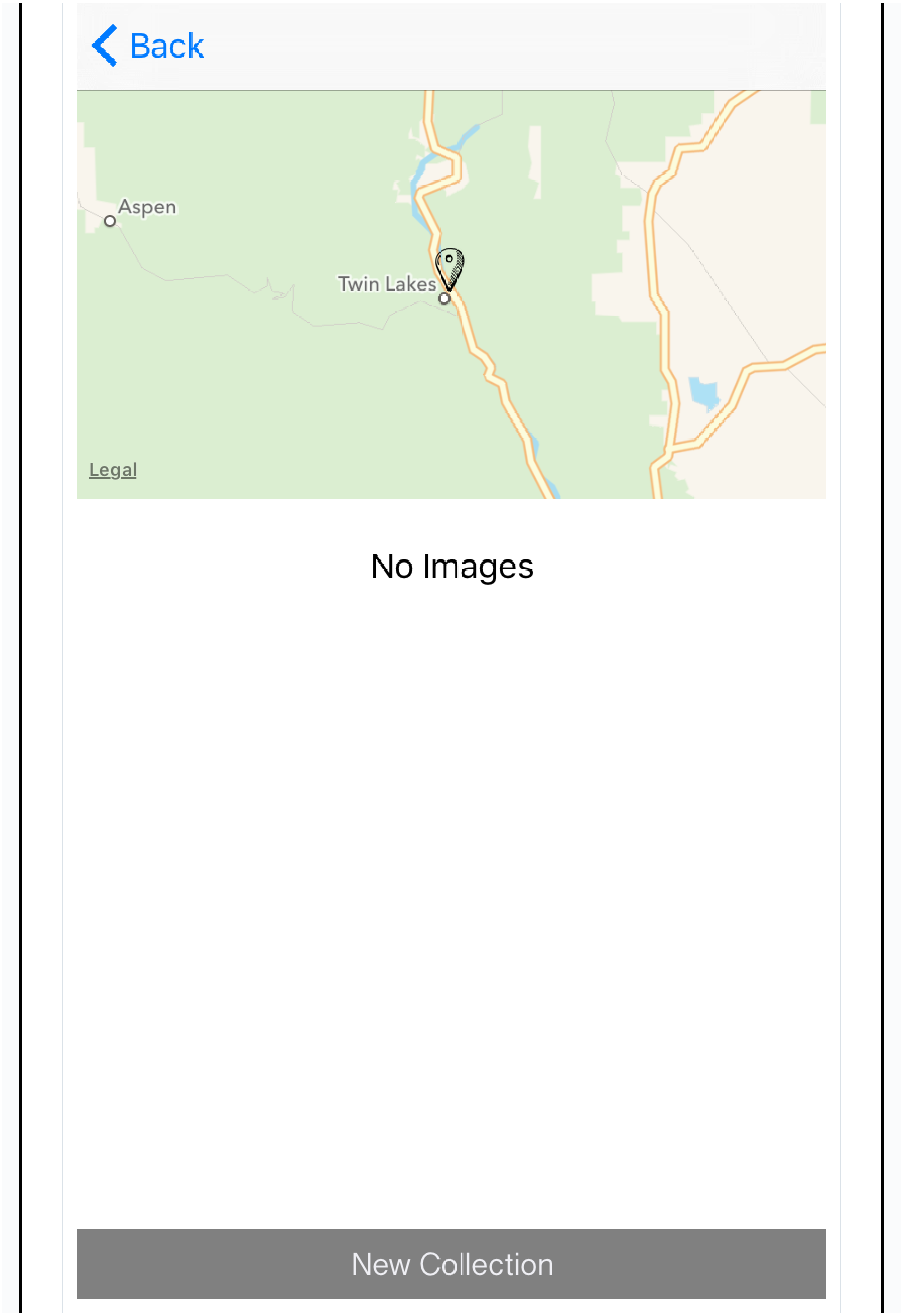
✓

When a photo is removed from an album, or when an entire album is deleted, the PhotoAlbumViewController explicitly deletes the underlying file(s) in the Documents directory.

Awesome job! Now, you should try exceeding specs here. CoreData includes a handy function that gives the oppurtunity to perform tasks right before an NSManagedObject is removed. prepareForDeletion() is a great place to house all your code for removing the underlying files stored in the documents directory and doing any additional cleanup you might have. You can read more about it in the Apple Documention for NSManagedObjects, but basically just add the function in your NSManagedObject class and fill it with the code you'd like to implement. The rest is managed for you! Neat huh? :)

The Photo Album view has a button that initiates the download of a new album, replacing the images in the photo album with a new set from Flickr.

Tapping the new collection button had unexpected results. It would delete some of the photos, the pressing it a few times more would delete all of the photos until the album was empty.


Back
Aspen
Twin Lakes
Legal
No Images
New Collection

Unfortunately, no new photos ever appeared.

✓

If the Photo Album view is opened for a pin that previously had photos assigned, they are immediately displayed. No new download is needed.

No wait here! Great work!

RESUBMIT

DOWNLOAD PROJECT

Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

Have a question about your review? Email us at review-support@udacity.com and include the link to this review.

RETURN TO PATH

Rate this review

★ ★ ★ ★ ★