**Case Study**

Name: Nirmohi Patel

Student ID: A00205892

1. PhotoStore.swift

* Inside fetchImage(for:completion) using guard statement to unwrap the optional NSURL. Then, bridge the NSURL instance to URL instance using an as cast. Now complier knows NSURL and URL are related. This is how model graph and photo entity defined
* To use core data framework, Import CoreData at the top
* Update fetchInterestingPhotos(completion:) with if case that can also return previous saved photos as well as newly inserted ones
* Describe fetch request entity so that object can know which entity to get. To fetch photo instance add new method fetchAllPhotos

1. FlikerAPI.swift

* Import CoreData at the top to manage object of their entity
* Update photo(fromJSON:) and add an argument NSManagedObjectContext to use context to new Photo instances. Add perfromAndWait() methd synchronous and return photo(fromJSON:into:) to interact with context on the queue that is already associated.
* Call photo(fromJSON:into:) method from photos(fromJSON:) NSManagedObjectContext and update processPhotosRequest method with passing viewContext tso that web service request complete successfully
* When app will run multiple time and same photos will return then it inserted into the context multiple time and to solve this error a unique identifier will be added that can identifier the same identifier multiple times. Update photo(fromJSON:into:) by checking ID of existing photos

1. PhotosViewController.swift

* Add new method to update data source with all photos updateDataSource
* Update viewDidLoad() method to fetch and display all photos saved to core data and previously saved photos will be returned when web service request get finished
* Update data source as soon as view is loaded