Core Data

Dong Duong,

April 2013

Course Objective

- Define Core Data
- How to use Core Data to persistence your data

Prerequisite

- Have knowledge with MVC
- Have knowledge Objective C (class, Instance variable, method, protocol)
- iOS Platform overview
- xCode and Interface Builder
- Or joined iOS overview course

Assessment Disciplines

- ❖ Class Participation : Required
- ❖ Assignment Completion : 100%
- ❖ Pass Score : >=70%

Course Timetable

- ❖ Lecture Duration + Hands-on Labs: 6 hours

Agenda

- History of Persistence in iOS
- What is Core Data.
- Core Data Basic
- Demo
- Working with Core Data
- Managing Table Views Using NSFetchedResultsController
- Practice
- Q&A

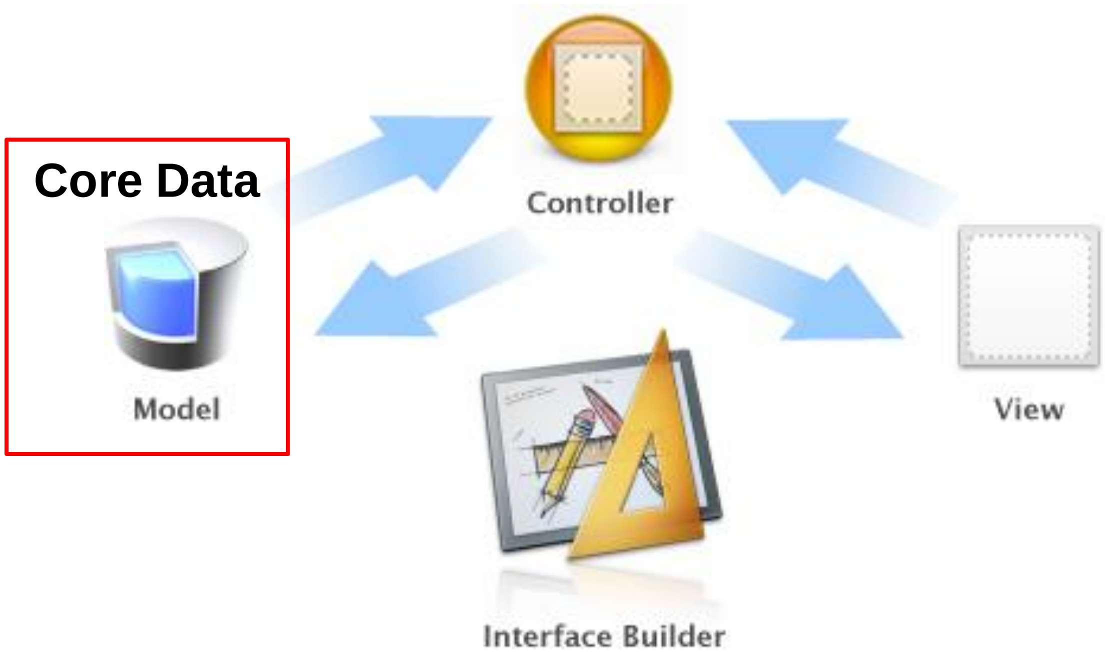
History of Persistence in iOS

- Use property lists, which contain nested lists of key/value pairs of various data types.
- Serialize objects to files using the SDK’s NSCoding protocol.
- Take advantage of the iPhone’s support for the relational database SQLite.
- Persist data to the Internet cloud.

What is Core Data?

What is Core Data?

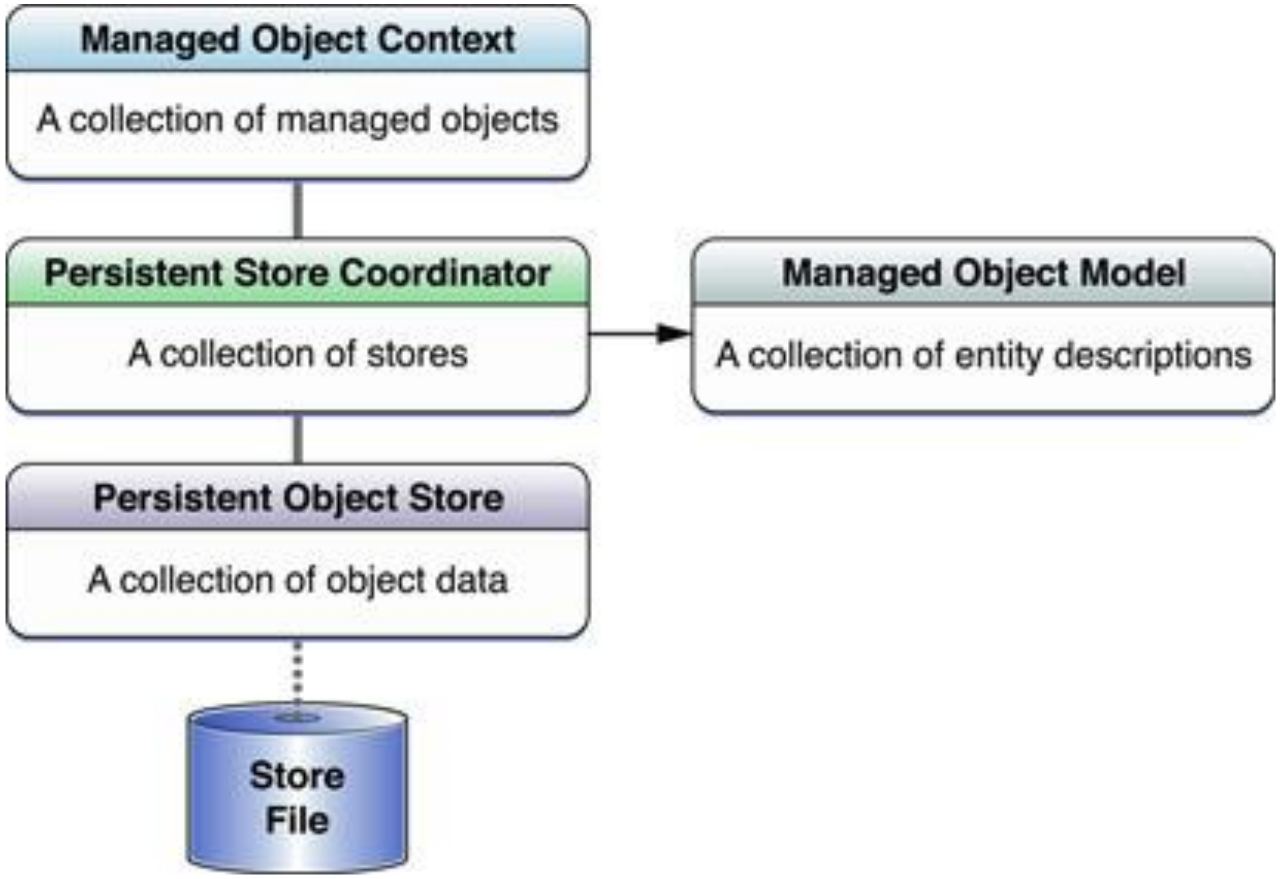
- Apple’s Core Data provides a versatile persistence framework.
- The Core Data framework supports the creation of model objects.
- Core Data hides most of the complexities of data storage.



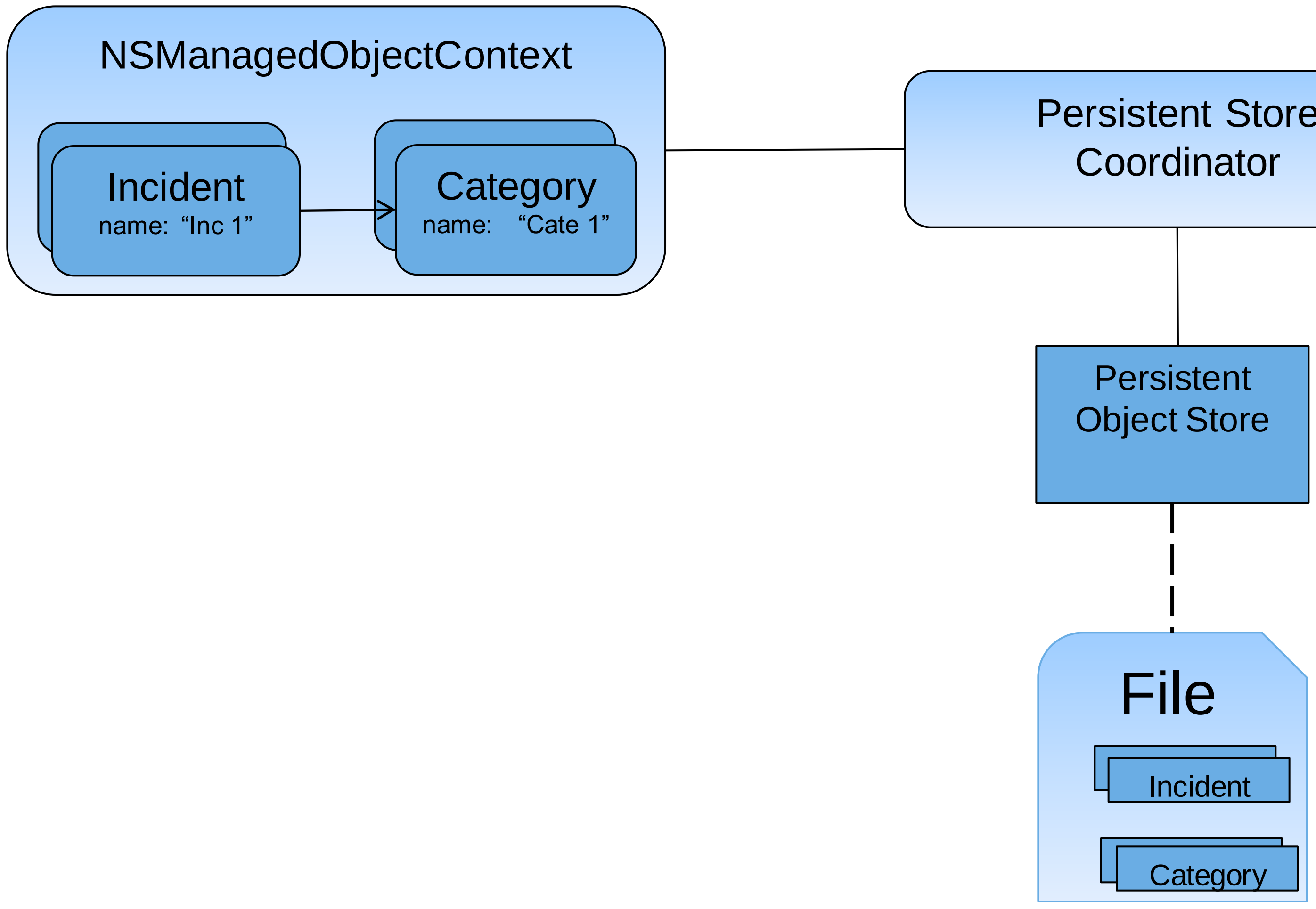
Core Data Basic

Basic Core Data Architecture

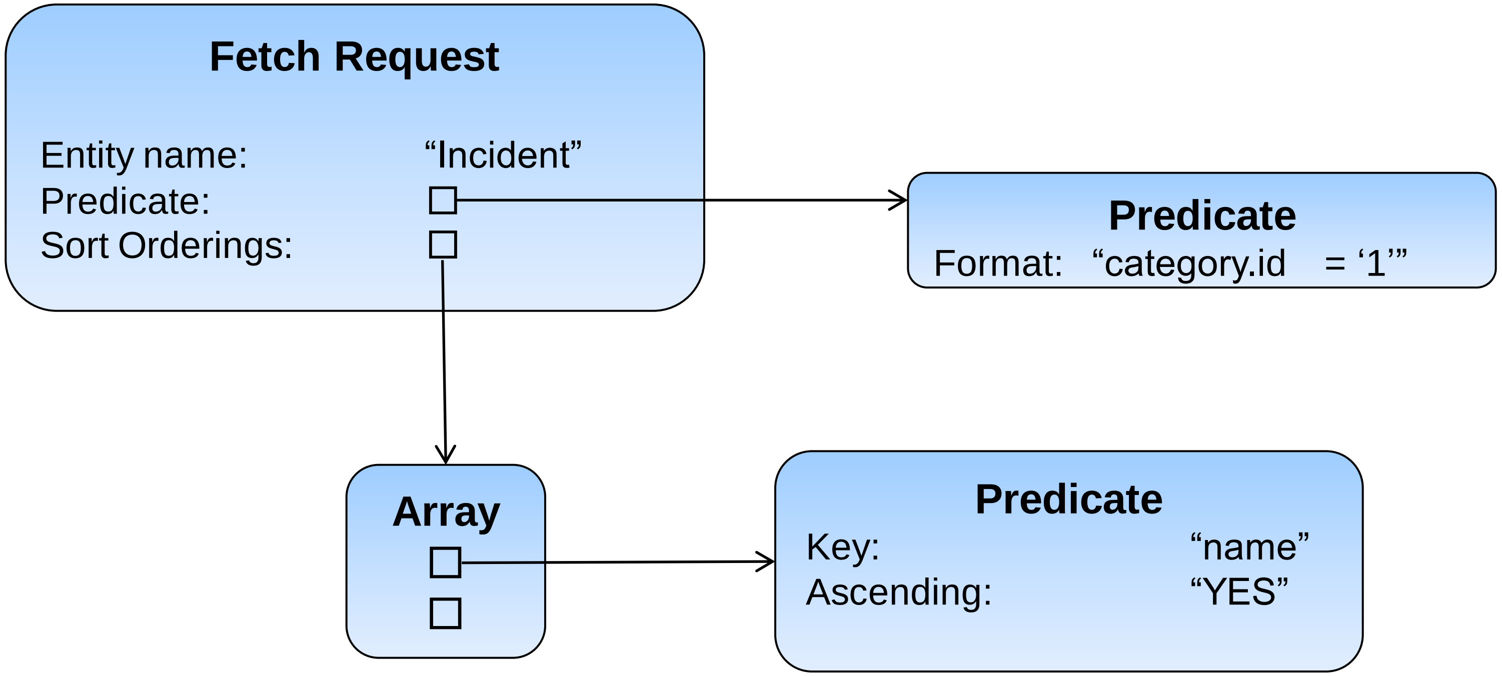
- Apple’s Core Data provides a versatile persistence framework.


Managed Object Context
A collection of managed objects
Persistent Store Coordinator
A collection of stores
Managed Object Model
A collection of entity descriptions
Persistent Object Store
A collection of object data
Store File

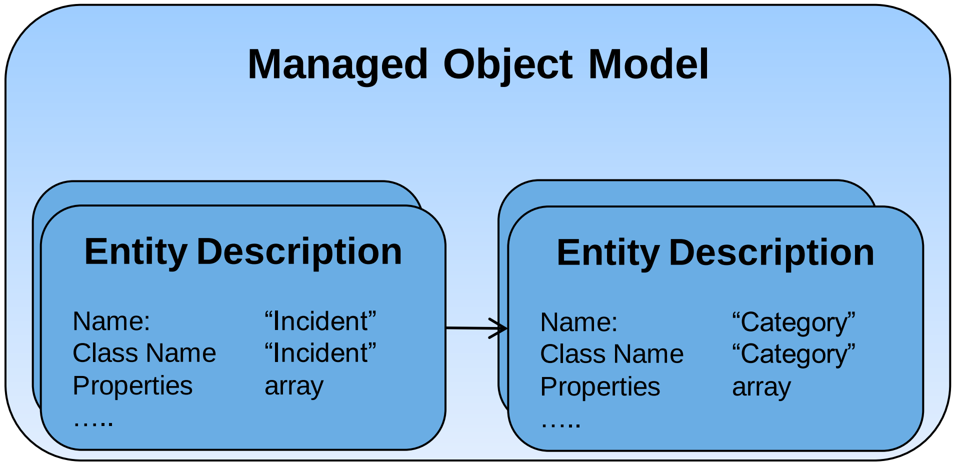
Managed Objects and the Managed Object Context

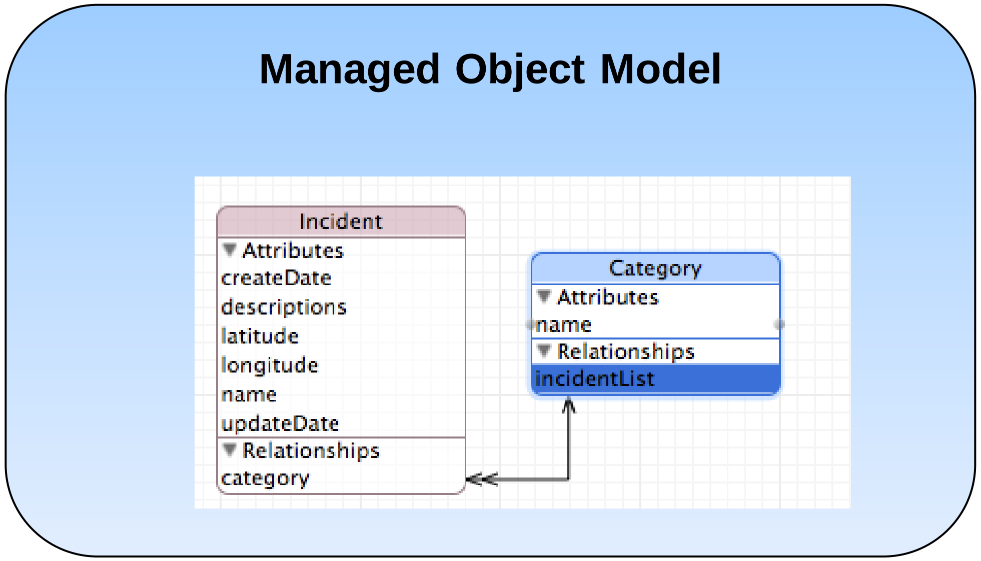


Fetch Requests



Managed Object Model





Demo: Create Core Data Model

Working with Core Data

Working with Core Data

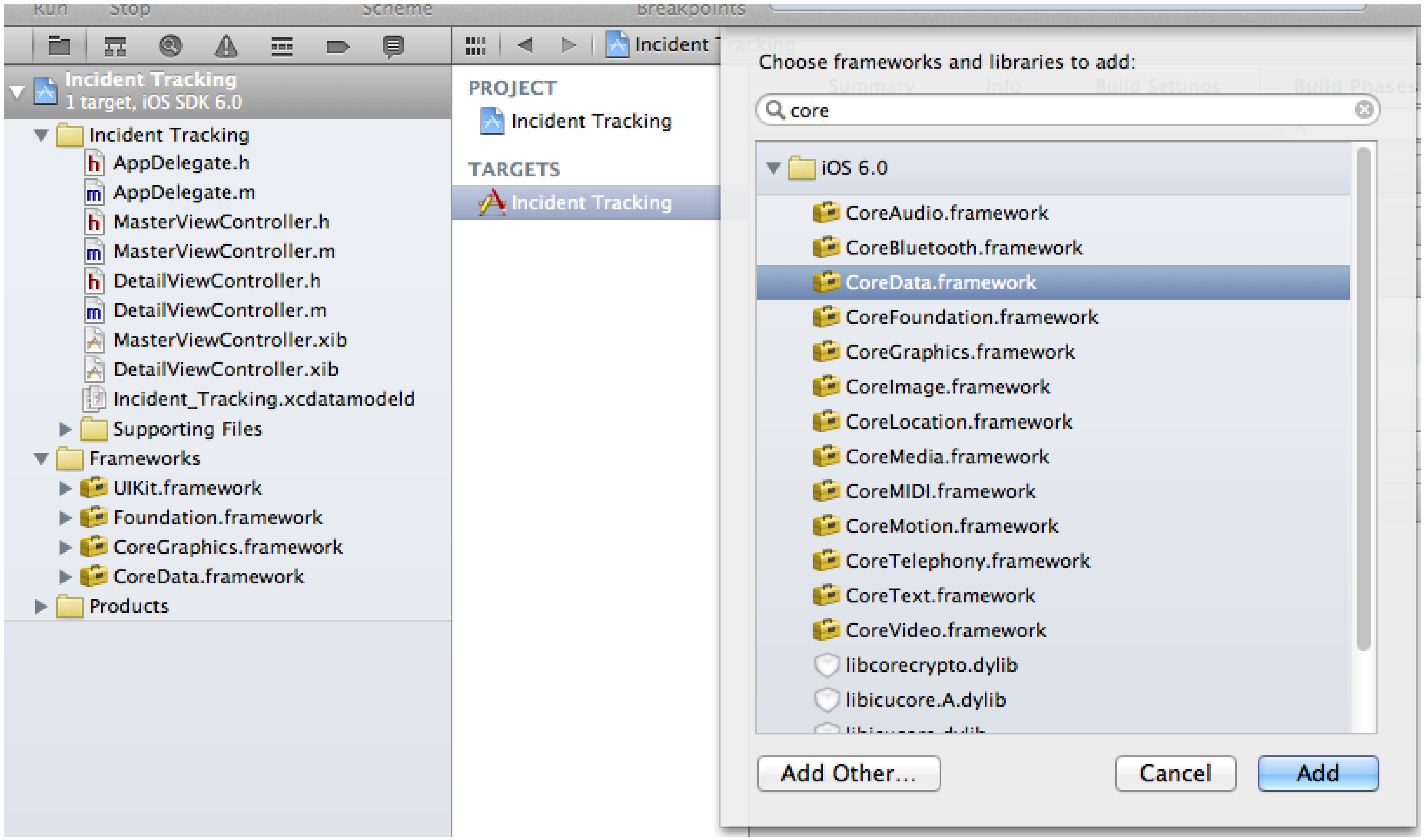
- Adding Core Data to existing project.
- Inserting new objects.
- Fetching Results.
- Deleting object.

Adding Core Data to existing project.

- Add the Core Data framework.
- Create a data model.
- Initialize the managed object context.

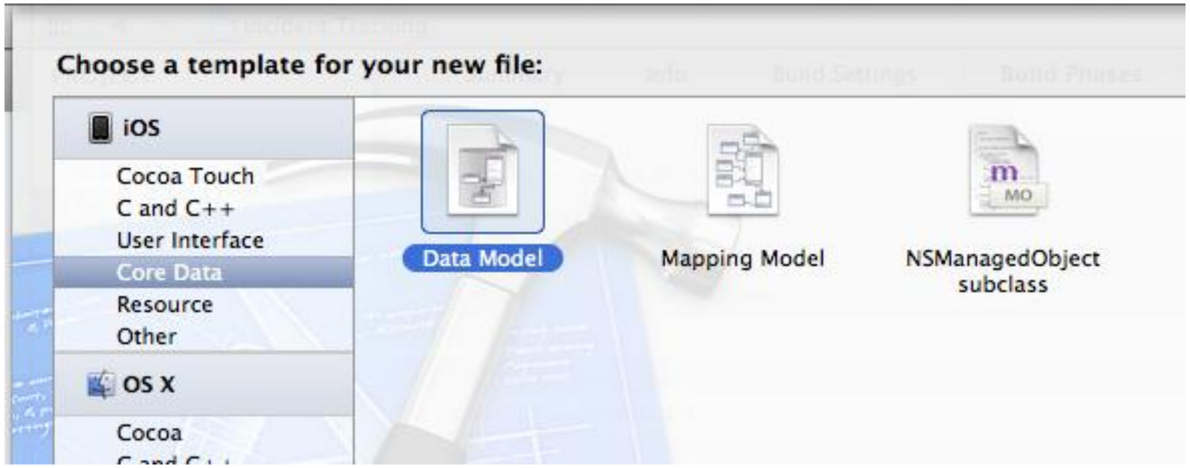
Adding Core Data to existing project.

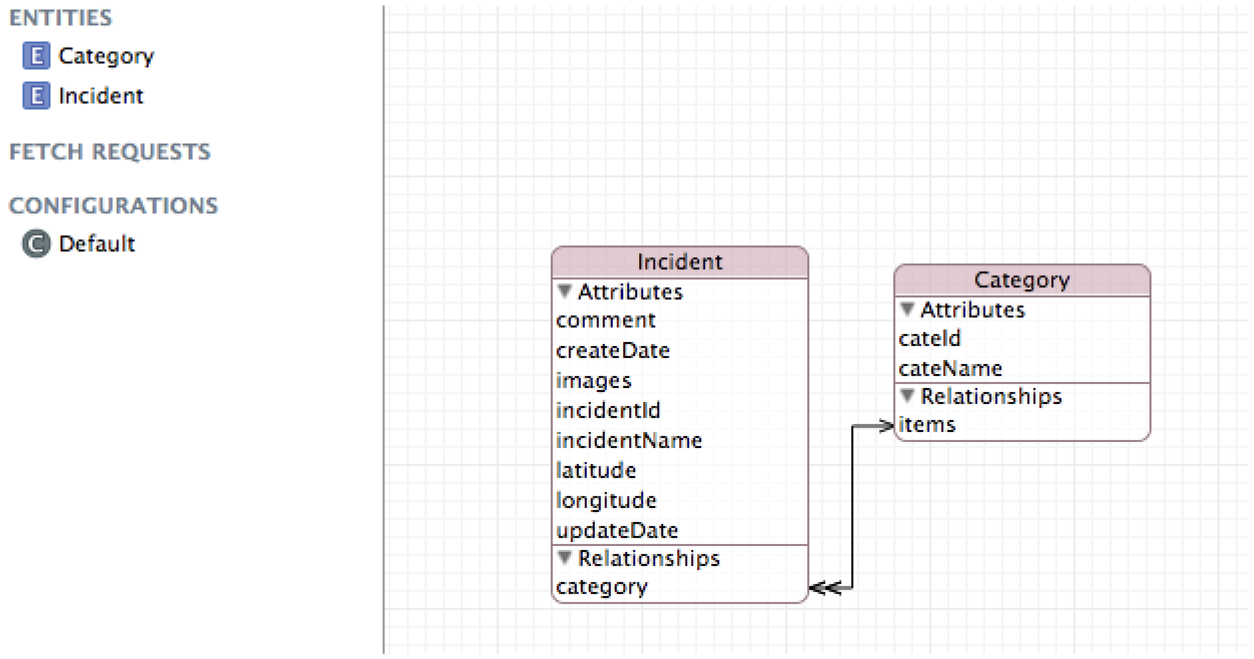
– Add the Core Data framework.



Adding Core Data to existing project.

– Create a data model. (File ➤ New File)





Initialize the managed object context.

» Adding properties to *AppDelegate.h*

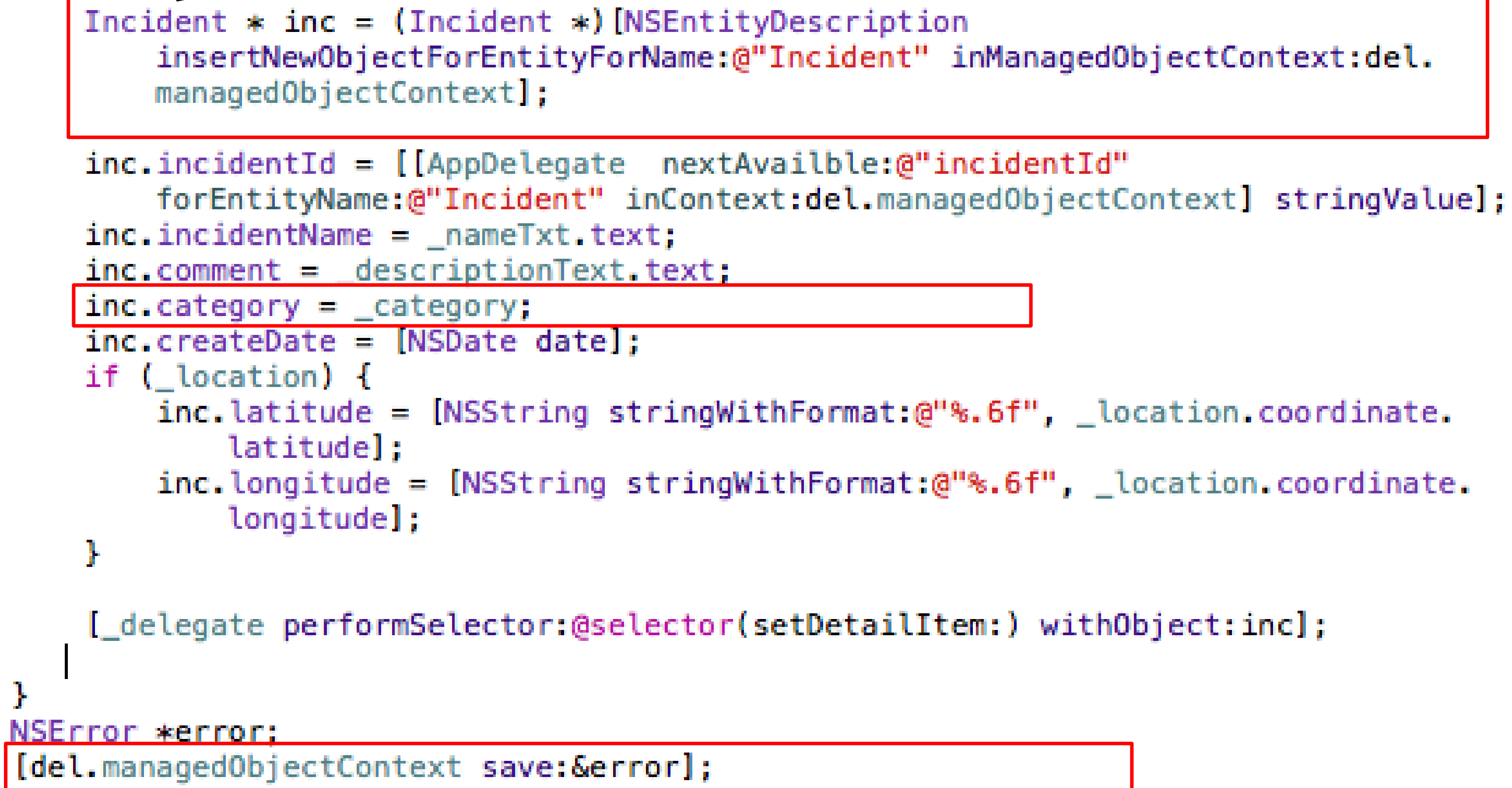
```
@interface AppDelegate : UIResponder <UIApplicationDelegate>

@property (strong, nonatomic) UIWindow *window;

@property (readonly, strong, nonatomic) NSManagedObjectContext *managedObjectContext;
@property (readonly, strong, nonatomic) NSManagedObjectModel *managedObjectModel;
@property (readonly, strong, nonatomic) NSPersistentStoreCoordinator *persistentStoreCoordinator;

- (void)saveContext;
```

– Implementing getter methods for properties in *AppDelegate.m*

```
// Returns the managed object context for the application.
// If the context doesn't already exist, it is created and bound to the persistent store
    coordinator for the application.
- (NSManagedObjectContext *)managedObjectContext
{
    if (_managedObjectContext != nil) {
        return _managedObjectContext;
    }

    NSPersistentStoreCoordinator *coordinator = [self persistentStoreCoordinator];
    if (coordinator != nil) {
        _managedObjectContext = [[NSManagedObjectContext alloc] init];
        [_managedObjectContext setPersistentStoreCoordinator:coordinator];
    }
    return _managedObjectContext;
}
```

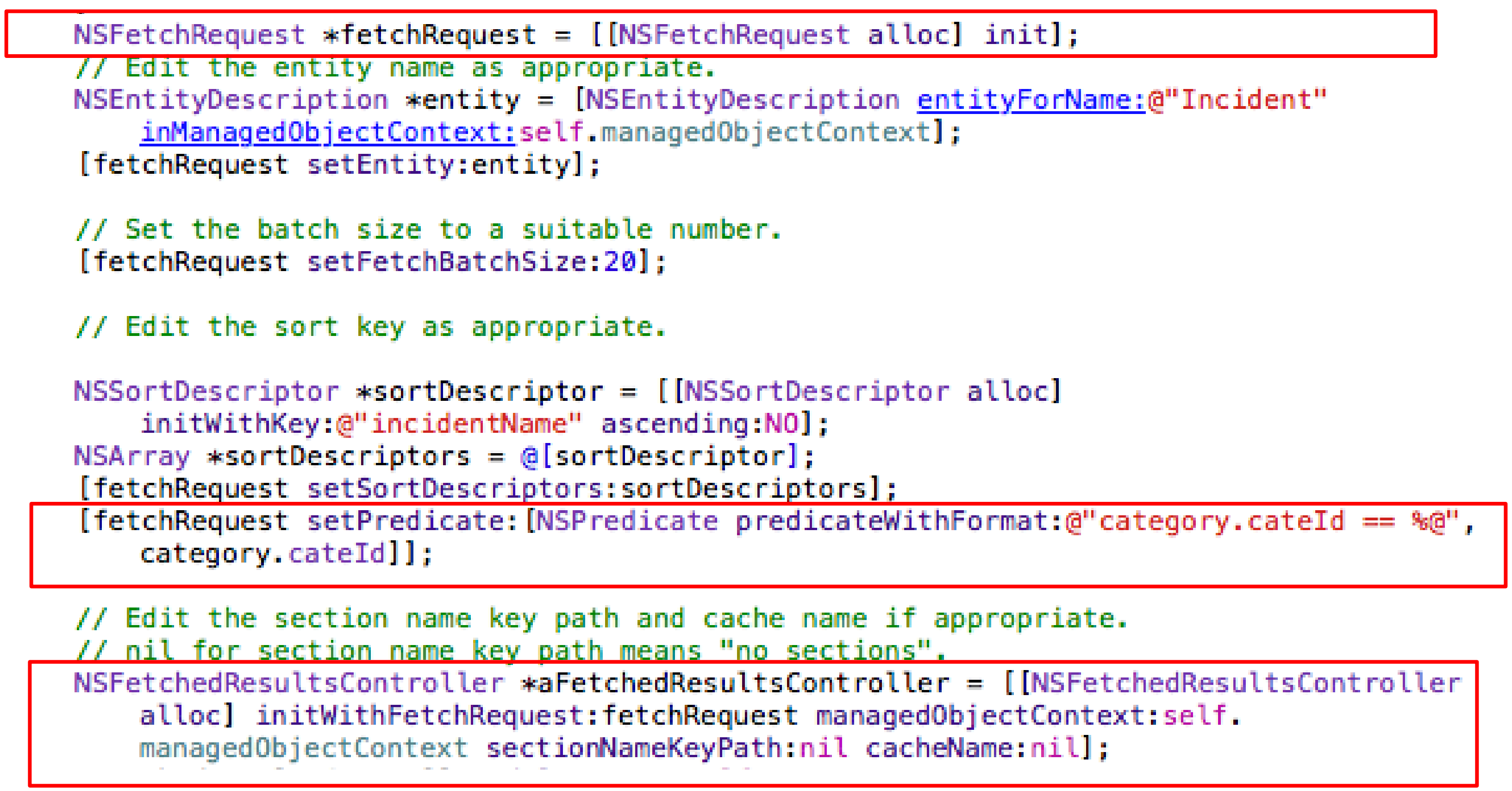
Inserting new objects

Model in MVC



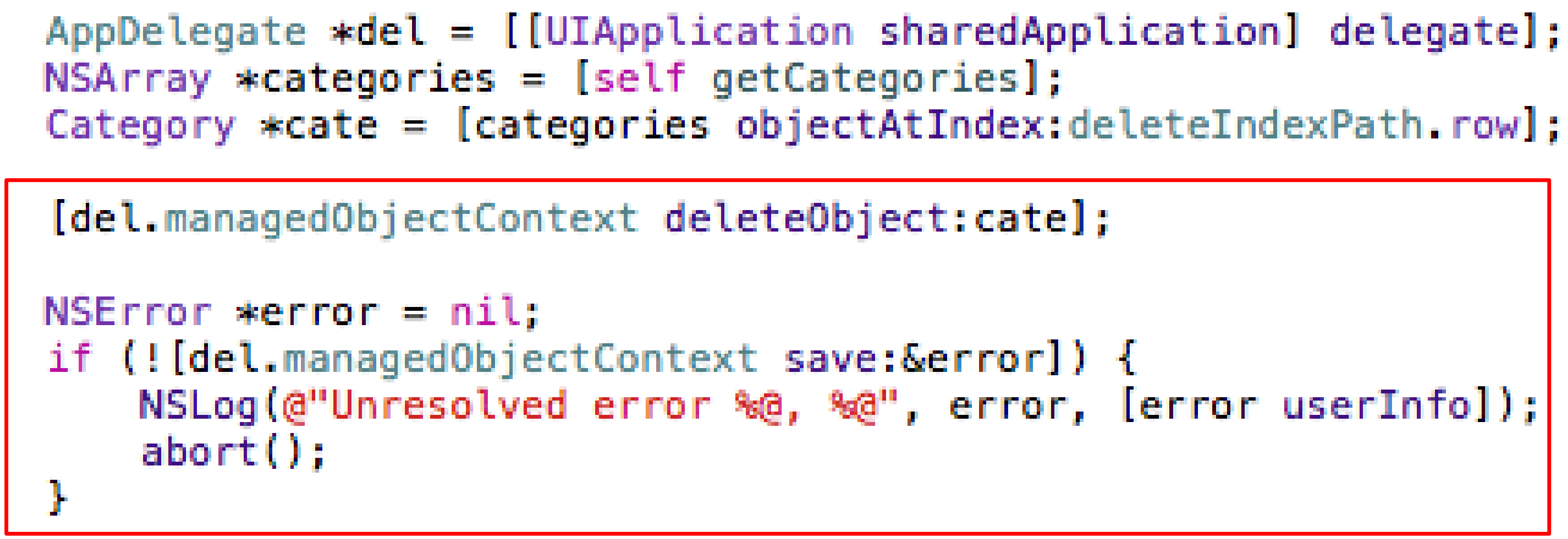
```
    Incident * inc = (Incident *)[NSEntityDescription
        insertNewObjectForEntityForName:@"Incident" inManagedObjectContext:del.
        managedObjectContext];

    inc.incidentId = [[AppDelegate  nextAvailble:@"incidentId"
        forEntityName:@"Incident" inContext:del.managedObjectContext] stringValue];
    inc.incidentName = _nameTxt.text;
    inc.comment = _descriptionText.text;
    inc.category = _category;
    inc.createDate = [NSDate date];
    if (_location) {
        inc.latitude = [NSString stringWithFormat:@"%.6f", _location.coordinate.
            latitude];
        inc.longitude = [NSString stringWithFormat:@"%.6f", _location.coordinate.
            longitude];
    }

    [_delegate performSelector:@selector(setDetailItem:) withObject:inc];

}
NSError *error;
[del.managedObjectContext save:&error];
```

Fetching Results.



```
NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];
// Edit the entity name as appropriate.
NSEntityDescription *entity = [NSEntityDescription entityForName:@"Incident"
    inManagedObjectContext:self.managedObjectContext];
[fetchRequest setEntity:entity];

// Set the batch size to a suitable number.
[fetchRequest setFetchBatchSize:20];

// Edit the sort key as appropriate.

NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc]
    initWithKey:@"incidentName" ascending:NO];
NSArray *sortDescriptors = @[sortDescriptor];
[fetchRequest setSortDescriptors:sortDescriptors];
[fetchRequest setPredicate:[NSPredicate predicateWithFormat:@"category.cateId == %@",
    category.cateId]];

// Edit the section name key path and cache name if appropriate.
// nil for section name key path means "no sections".
NSFetchedResultsController *aFetchedResultsController = [[NSFetchedResultsController
    alloc] initWithFetchRequest:fetchRequest managedObjectContext:self.
    managedObjectContext sectionNameKeyPath:nil cacheName:nil];
```

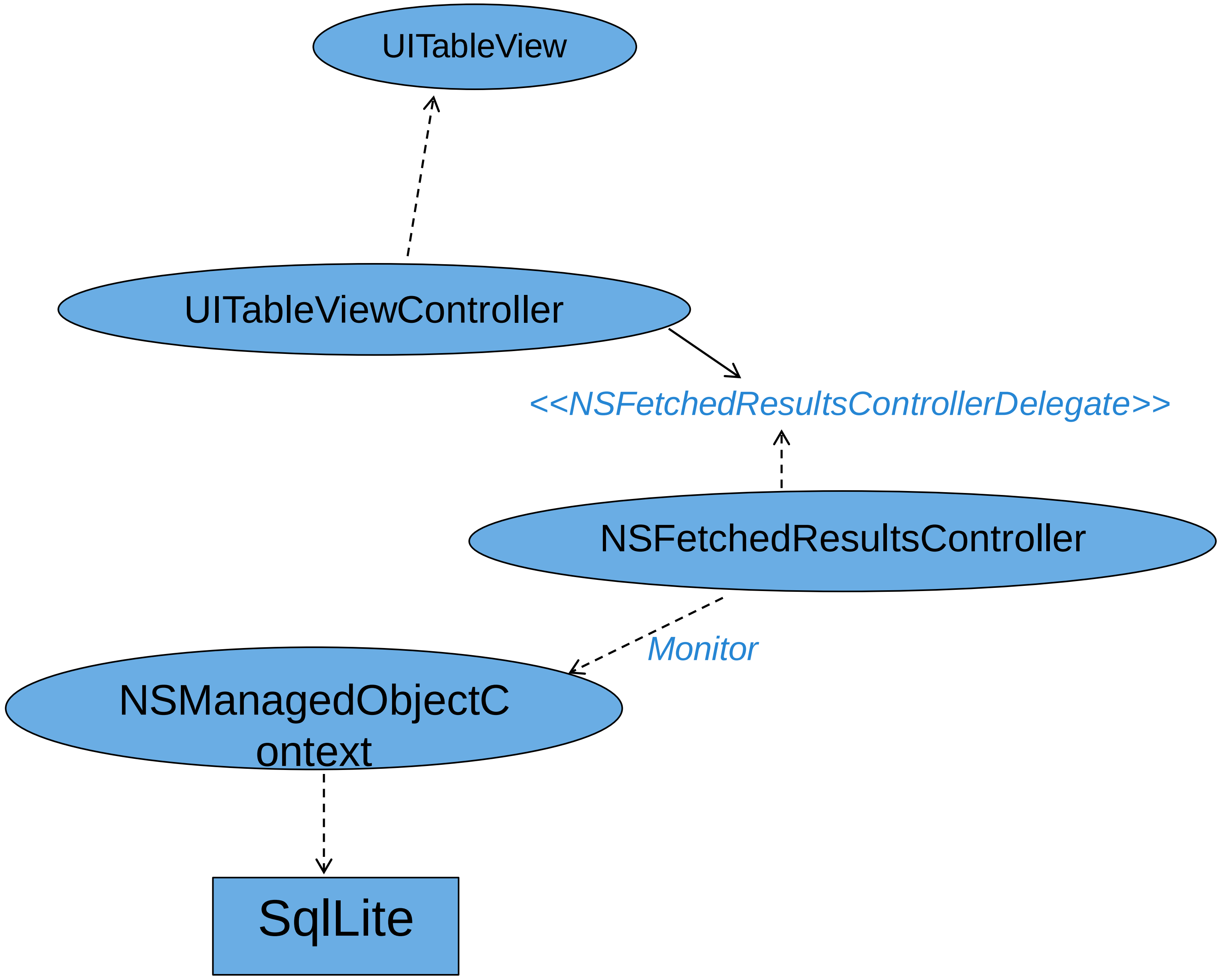
Deleting object



```
AppDelegate *del = [[UIApplication sharedApplication] delegate];
NSArray *categories = [self getCategories];
Category *cate = [categories objectAtIndex:deleteIndexPath.row];

[del.managedObjectContext deleteObject:cate];

NSError *error = nil;
if (![del.managedObjectContext save:&error]) {
    NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
    abort();
}
```

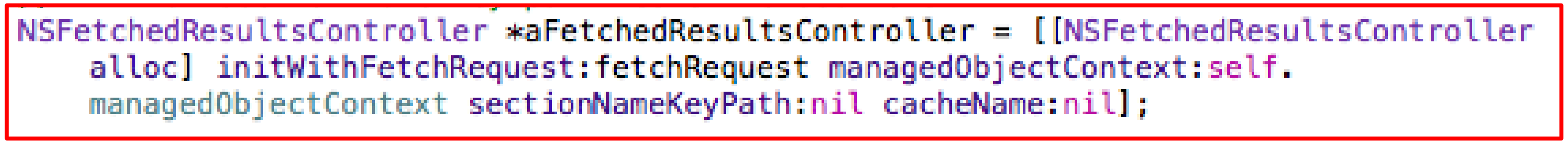
Practice 1: Adding Core Data to Incident Tracking Project

Managing Table Views Using NSFetchedResults Controller


UITableView
UITableViewController
<<NSFetchedResultsControllerDelegate>>
NSFetchedResultsController
Monitor
NSManagedObjectC
ontext
SqlLite

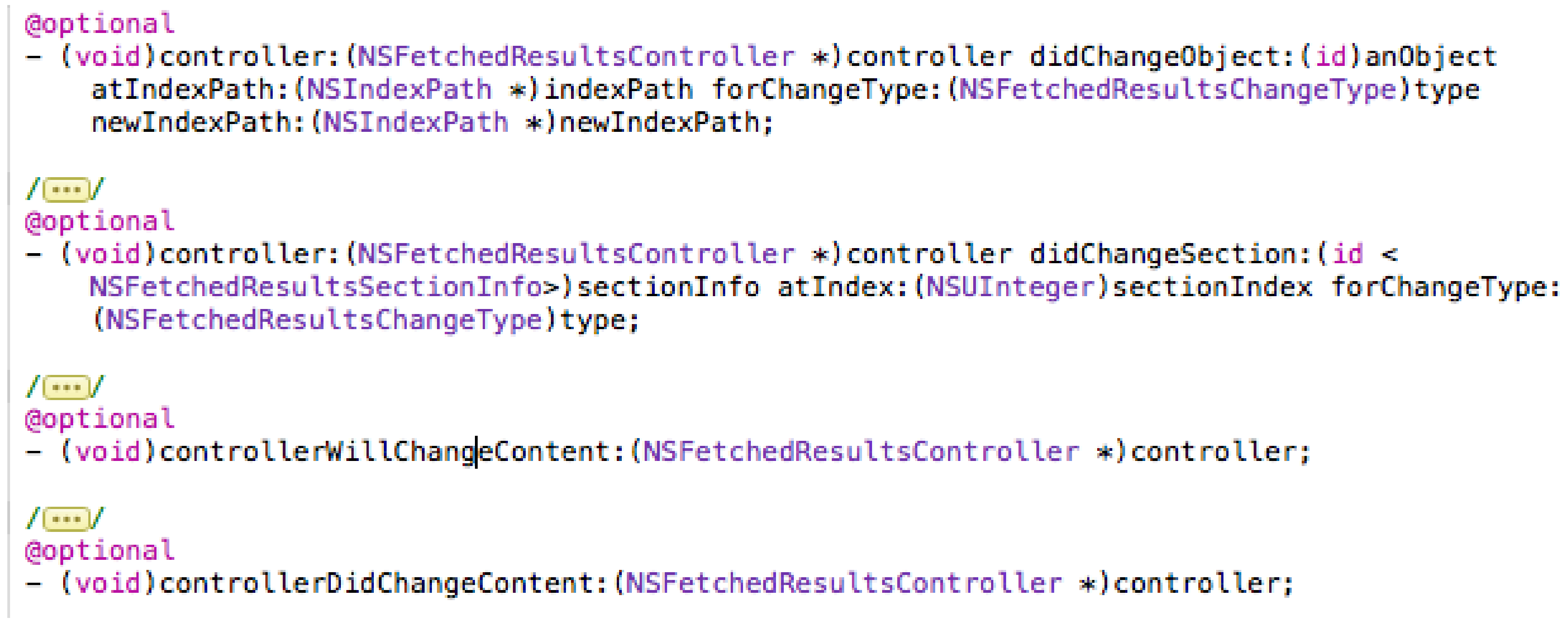
NSFetchedResultsController

- Works closely with UITableView instances to display data from a Core Data data model in a table view.
- It also manages adding, removing, and moving rows in the table in response to data changes.
- You create a fetched results controller with four parameters:
  - A fetch request (NSFetchRequest instance)
  - A managed object context
  - A section name key path
  - A cache name



```
NSFetchedResultsController *aFetchedResultsController = [[NSFetchedResultsController
    alloc] initWithFetchRequest:fetchRequest managedObjectContext:self.
    managedObjectContext sectionNameKeyPath:nil cacheName:nil];
```

- NSFetchedResultsController Delegates (Controller in MVC)



```
@optional
- (void)controller:(NSFetchedResultsController *)controller didChangeObject:(id)anObject
    atIndexPath:(NSIndexPath *)indexPath forChangeType:(NSFetchedResultsChangeType)type
    newIndexPath:(NSIndexPath *)newIndexPath;

/•••/
@optional
- (void)controller:(NSFetchedResultsController *)controller didChangeSection:(id <
    NSFetchedResultsSectionInfo>)sectionInfo atIndex:(NSUInteger)sectionIndex forChangeType:
    (NSFetchedResultsChangeType)type;

/•••/
@optional
- (void)controllerWillChangeContent:(NSFetchedResultsController *)controller;

/•••/
@optional
- (void)controllerDidChangeContent:(NSFetchedResultsController *)controller;
```

Practice 2: Using core data to store the incident object.

THANK YOU

CSC

BUSINESS SOLUTIONS
TECHNOLOGY
OUTSOURCING