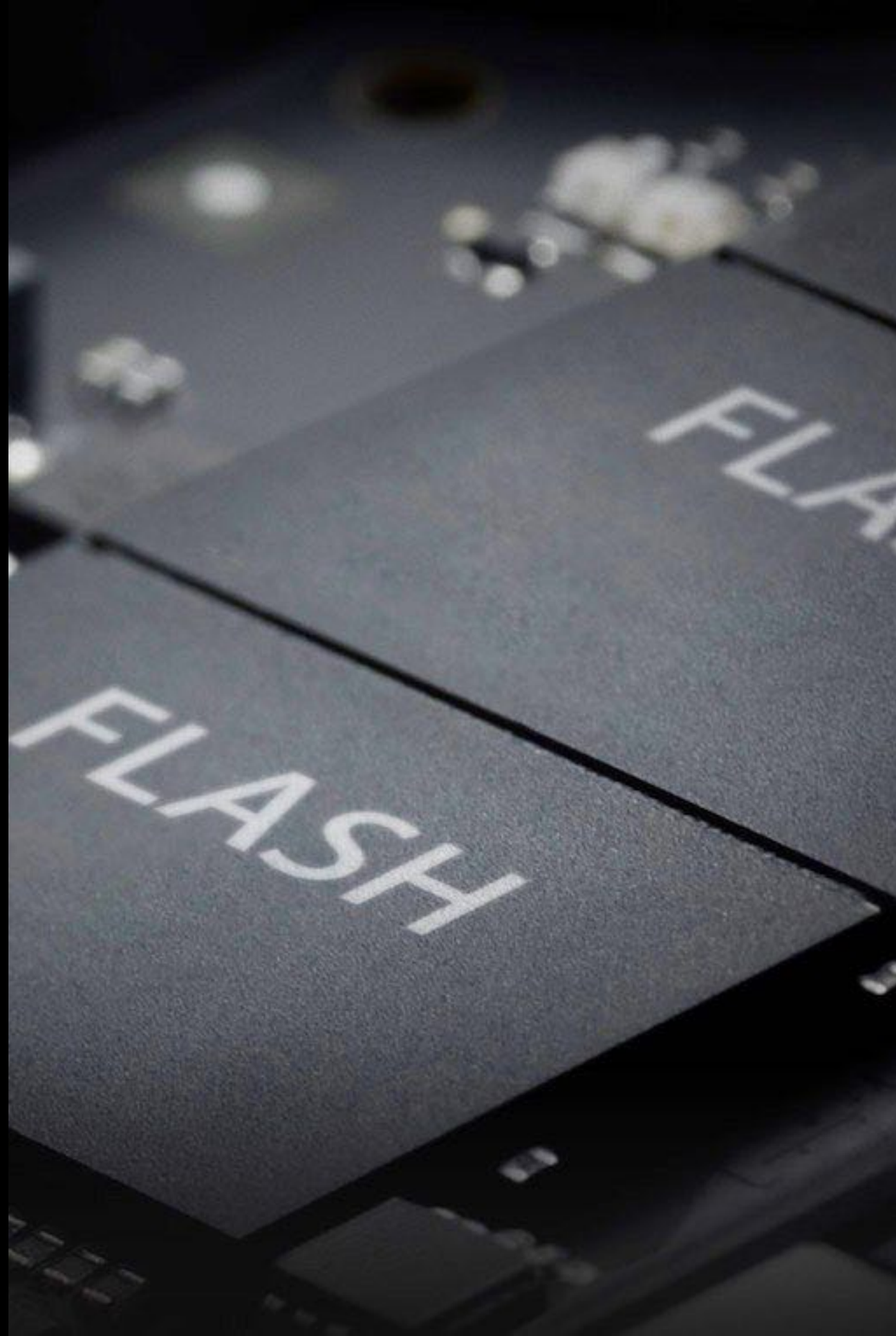
LUKASZ KAWKA PRESENTING

PERSISTANCE IN IOS

3 WAYS TO GO

- Simple persistence with User Defaults
- Sandboxing
- Advanced persistence with Core Data



A KEY-VALUE STORE

USER DEFAULTS

- Good place for storing preferences
- Can store only basic types like String, Bool, Int…
- Don’t store sensitive data there
- Don’t use it for large amounts of data
- TIP: If you want to set default values do it in the app delegate

SAMPLE

```
if UserDefaults.standard.value(forKey: "AnimateTables") == nil {
    UserDefaults.standard.set(true, forKey: "AnimateTables")
}
```

SAVING FILES TO THE FILE SYSTEM

SANDBOXING

- Best way to store files like images or documents

SAMPLE

```
func saveImageLocally(id: String) -> String {
        let documentDirectory = try! FileManager.default.url(for: .documentDirectory, in: .userDomainMask, appropriateFor:
nil, create: true)

        let imageURL = documentDirectory.appendingPathComponent(“images\(id).png")

        do {
            try UIImageJPEGRepresentation(self, 0.5)!.write(to: imageURL)
        } catch {
            print("error with saving image")
            return ""
        }
        return "images\(id).jpg"
    }

    func getImage(forPath path: String?) -> UIImage? {
        guard let path = path, path != "" else {return nil}

        let imageURL = try! FileManager.default.url(for: .documentDirectory, in: .userDomainMask, appropriateFor: nil, create:
true).appendingPathComponent(path)

        return UIImage(contentsOfFile: imageURL.path)
    }
```

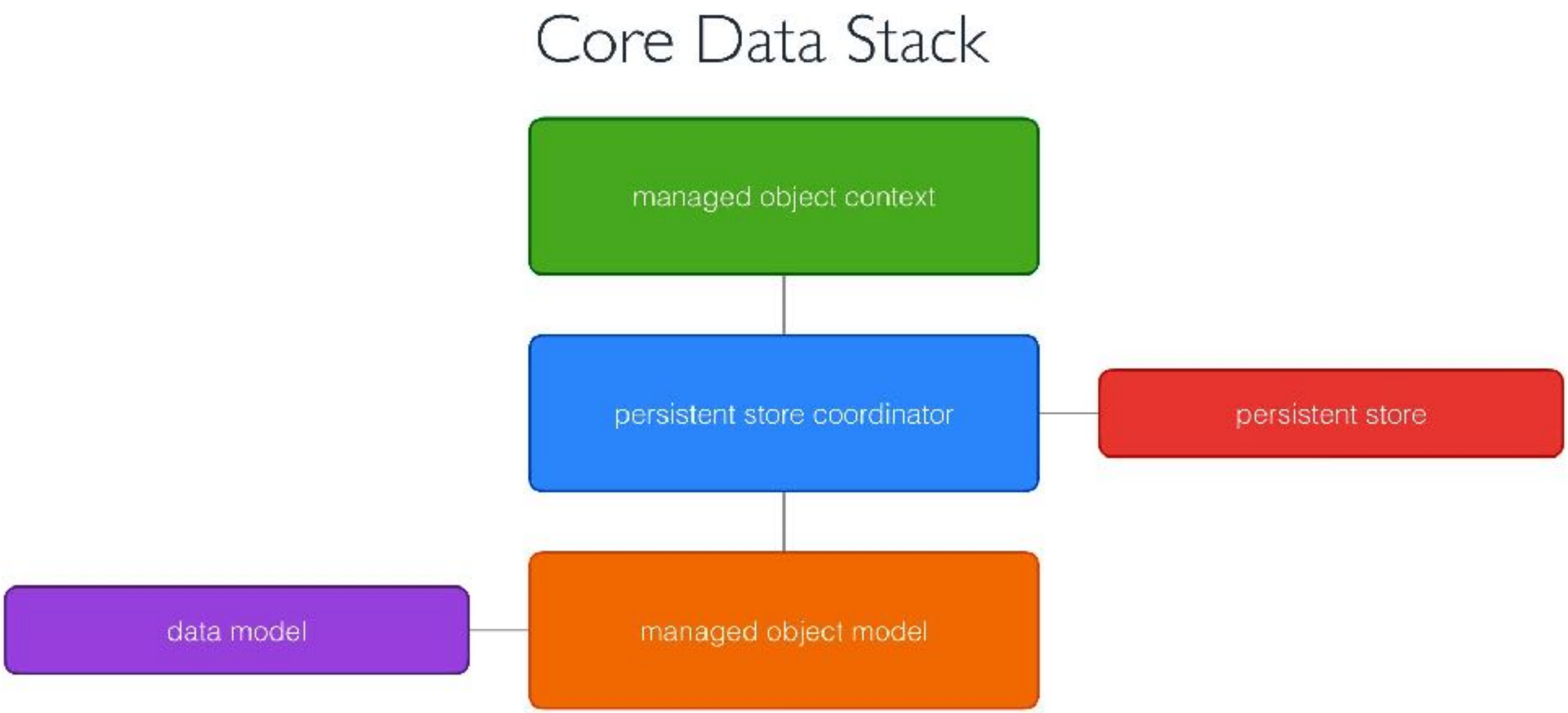
DATABASE PERSISTANCE

CORE DATA

WHY CORE DATA?

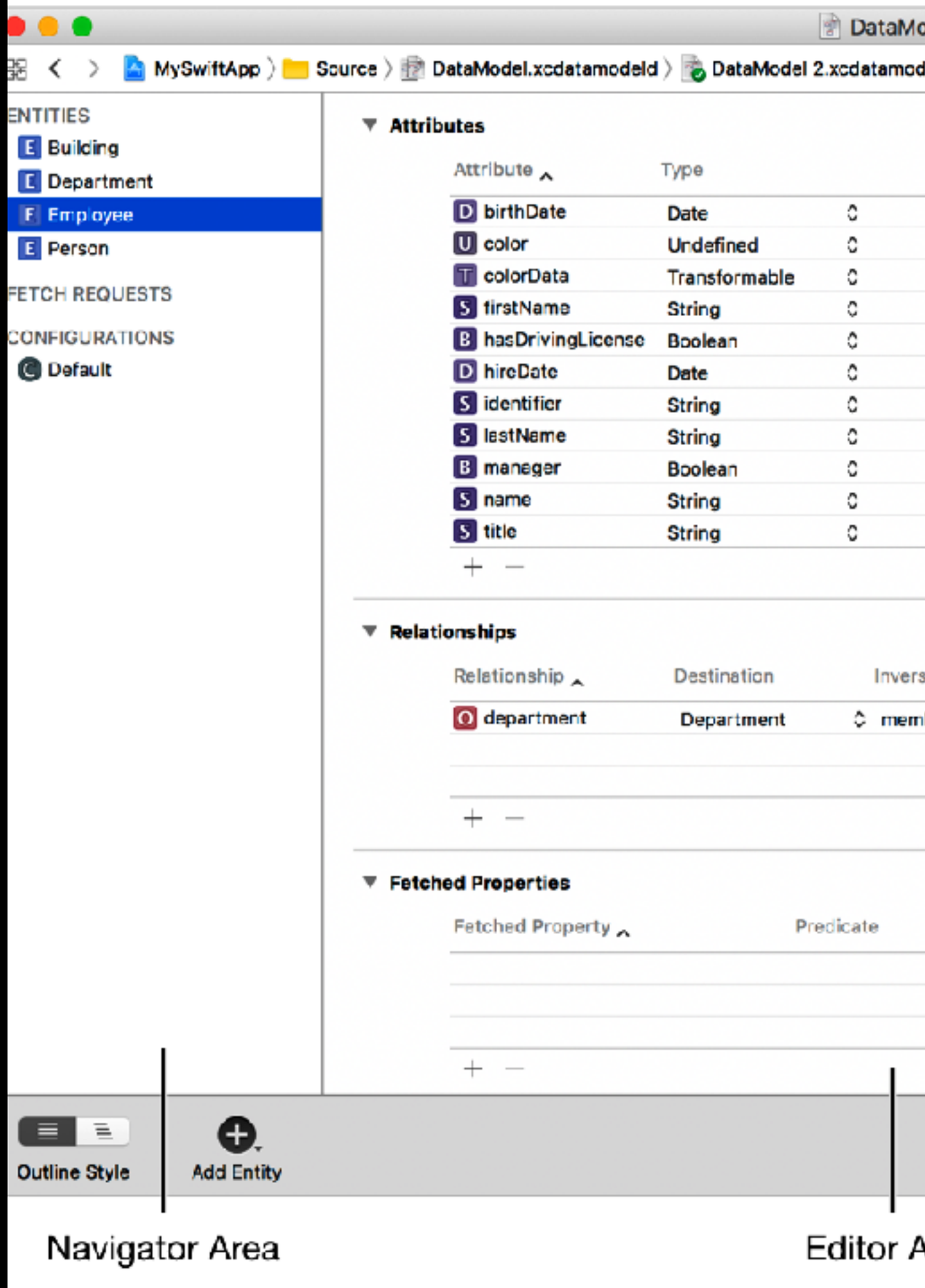
- Change tracking and built-in management of undo and redo beyond basic text editing
- Core Data and related classes provide easy ways to get your entities into UITableViews, like NSFetchedResultsController
- Core Data abstracts away a lot of the messy things you'd otherwise have to deal with yourself, such as lists of objects, one-to-many or many-to-many relationships
- Core Data comes with a nice graphical object model editor that can help you think through your object/entity design, and refine it as you go.
- More reasons
- You can also directly use SQLite or third party databases and frameworks like Firebase

OVERVIEW


Core Data Stack
managed object context
persistent store coordinator
persistent store
data model
managed object model

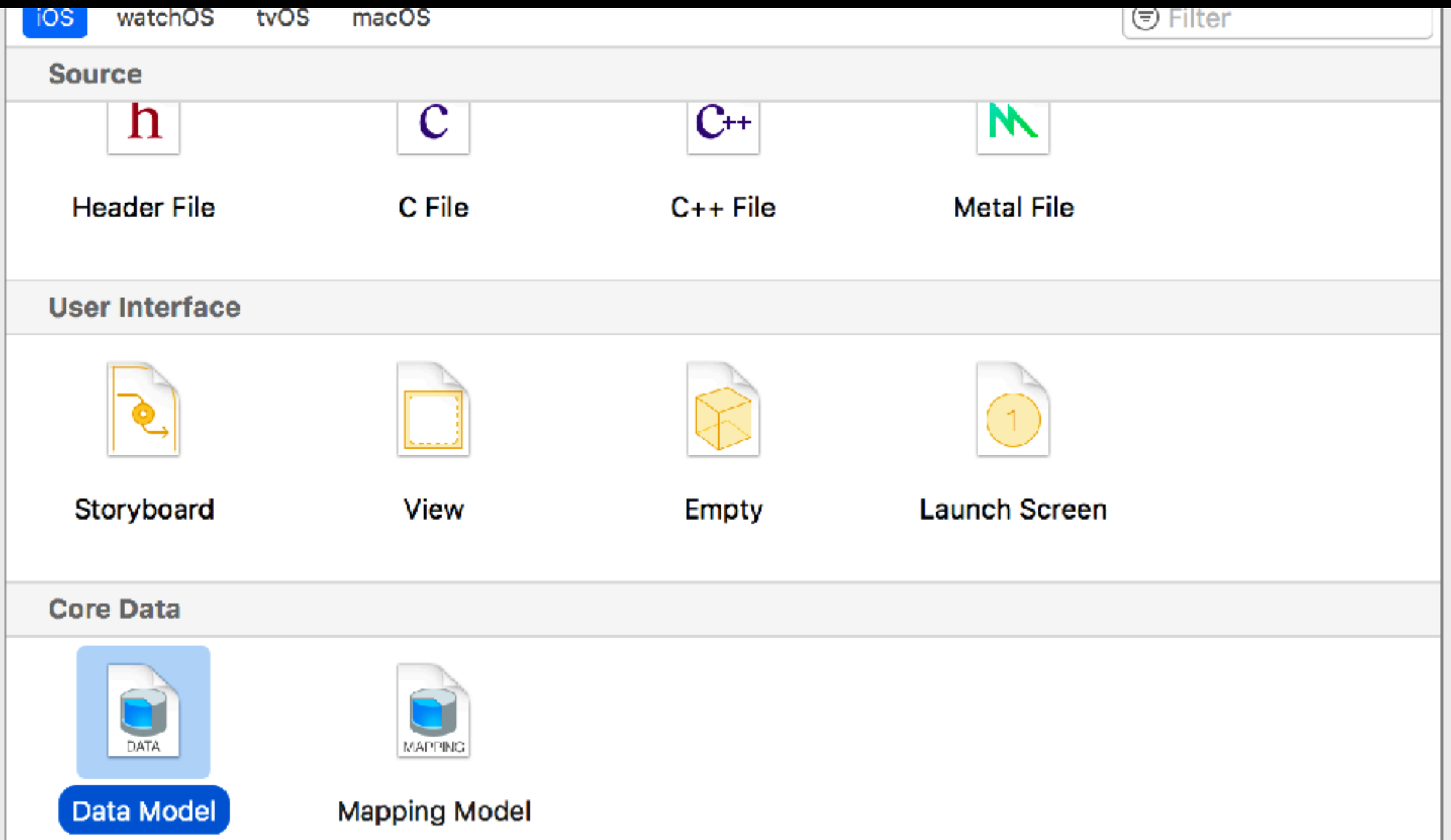
DATA MODEL

- Create a model
- Create and setup entities
- Setup relationships
- Subclass if you need to



NOTHING SPECIAL

CREATE A MODEL

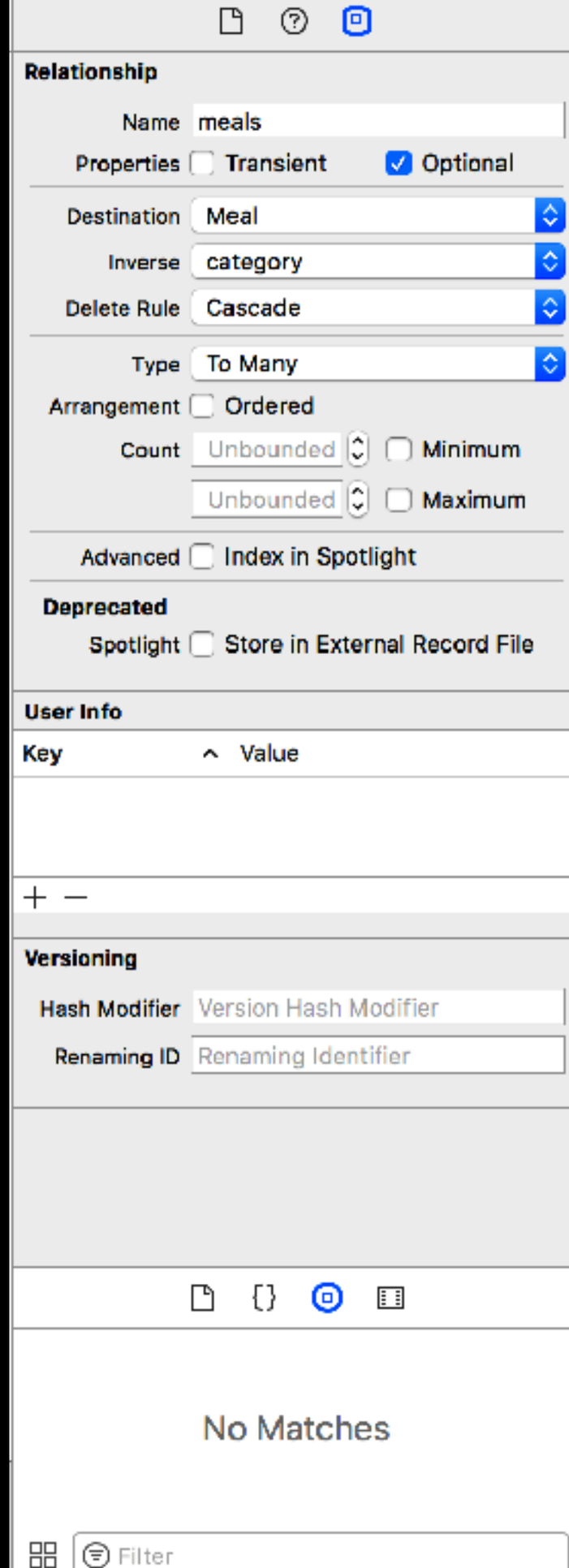


ENTITIES

- Think of entity as a class
- And attributes as variables
- Attributes must be of a certain supported type like Double or Binary Data
- You can store larger files like images as Binary Data, but it’s not recommended since Core Data works best on large amount of small pieces of data

RELATIONSHIPS

- Types:
  - One-to-one e.g. one owner for one notebook
  - One-to-many e.g. one notebook for many notes
  - Many-to-many e.g. many categories for many notebooks
- Deletion rules:
  - No action
  - Nullify - the destination of the relationship is nullified when the record is deleted
  - Cascade - deletion of all associated objects
  - Deny - prevents the deletion of the record



CORE DATA STACK

- Persistance Container
- Managed object model
- Persistence store coordinator
- Managed object context

PERSISTANCE CONTAINER

- Handles the creation of the Core Data stack
- Introduced in iOS 10

MANAGED OBJECT MODEL

- Describes the data that is going to be accessed by the Core Data stack
- Is loaded into memory as the first step in the creation of the stack
- After the NSManagedObjectModel object is initialized, the NSPersistentStoreCoordinator object is constructed

PERSISTENCE STORE COORDINATOR

- Realizes instances of entities that are defined inside of the model. It creates new instances of the entities in the model, and it retrieves existing instances from a persistent store

- NSManagedObjectModel defines the structure of the data, the NSPersistentStoreCoordinator realizes objects from the data in the persistent store and passes those objects off to the requesting NSManagedObjectContext.

- Verifies that the data is in a consistent state that matches the definitions in the NSManagedObjectModel.

- The call to add the NSPersistentStore to the NSPersistentStoreCoordinator is performed asynchronously

MANAGED OBJECT CONTEXT

- Object that your application will interact with the most, and therefore it is the one that is exposed to the rest of your application
- Think of it as an intelligent scratch pad. You bring temporary copies onto the scratch pad where they form a collection of object graphs. Unless you actually save those changes, however, the persistent store remains unaltered
- Saving:

```
do {
    try managedObjectContext.save()
} catch {
    fatalError("Failure to save context: \(error)")
}
```

SUBCLASSING MANAGED OBJECT

- How to:
  - In the model select the entity you want to subclass
  - Than in Data Model Inspector (in Utilities) change the Codegen property to Manual/None
  - Go to Editor -> Create NSManagedObject Subclass… and follow the steps
- It should create 2 files for every entity you picked
- Why would I do this? Here is an example of custom init:

```
convenience init(name: String, context: NSManagedObjectContext) {
    if let entity = NSEntityDescription.entity(forEntityName: "Meal", in: context) {
        self.init(entity: entity, insertInto: context)

        self.name = name
    } else {
        fatalError("Unable to find Category entity name")
    }
}
```

FETCHING REQUEST

```
let fetchRequest = NSFetchRequest<NSFetchRequestResult>(entityName: "Meal")
fetchRequest.sortDescriptors = [NSSortDescriptor(key: "name", ascending: true)]
fetchRequest.predicate = NSPredicate(format: "name = %@", argumentArray: [name])
```

```
do {
    let meals = try context.fetch(fetchRequest) as? [Meal]
} catch {
    print("Failed to fetch meals: \(error)")
}
```

More about NSPredicate:
https://academy.realm.io/posts/nspredicate-cheatsheet/

FETCHED RESULTS CONTROLLER

- Nicely integrates with UIKit
- Has delegate class with some very useful methods

FETCHED RESULTS CONTROLLER

```
let mealsFRC =
NSFetchedResultsController(fetchRequest: fetchRequest,
managedObjectContext: stack.context,
sectionNameKeyPath: nil, cacheName: nil)

if let meals = mealsFRC.fetchedObjects as? [Meal] {}
```

FETCHED RESULTS CONTROLLER DELEGATE

```
func controllerWillChangeContent(_ controller:
NSFetchedResultsController<NSFetchRequestResult>) {
        self.beginUpdates()
    }

func controller(_ controller: NSFetchedResultsController<NSFetchRequestResult>, didChange
sectionInfo: NSFetchedResultsSectionInfo, atSectionIndex sectionIndex: Int, for type:
NSFetchedResultsChangeType) {

    let set = IndexSet(integer: sectionIndex)

    switch (type) {
    case .insert:
        self.insertSections(set, with: .fade)
    case .delete:
        self.deleteSections(set, with: .fade)
    default:
        break
    }
}
```

FETCHED RESULTS CONTROLLER DELEGATE

```
func controller(_ controller: NSFetchedResultsController<NSFetchRequestResult>, didChange anObject: Any, at indexPath: IndexPath?, for type: NSFetchedResultsChangeType, newIndexPath: IndexPath?) {

    switch(type) {
    case .insert:
        self.insertRows(at: [newIndexPath!], with: .fade)
    case .delete:
        self.deleteRows(at: [indexPath!], with: .fade)
    case .update:
        self.reloadRows(at: [indexPath!], with: .fade)
    case .move:
        self.deleteRows(at: [indexPath!], with: .fade)
        self.insertRows(at: [newIndexPath!], with: .fade)
    }
}

func controllerDidChangeContent(_ controller: NSFetchedResultsController<NSFetchRequestResult>) {
    self.endUpdates()
}
```

SAVING CONTEXT

```
func autoSave(_ delayInSeconds : Int) {
    if delayInSeconds > 0 {
        do {
            try saveContext()
            print("Autosaving")
        } catch {
            print("Error while autosaving")
        }

        let delayInNanoSeconds = UInt64(delayInSeconds) * NSEC_PER_SEC
        let time = DispatchTime.now() + Double(Int64(delayInNanoSeconds)) / Double(NSEC_PER_SEC)

        DispatchQueue.main.asyncAfter(deadline: time) {
            self.autoSave(delayInSeconds)
        }
    }
}
```

- Multiple contexts tutorial (might be useful when you do background tasks..): https://www.raywenderlich.com/174082/multiple-managed-object-contexts-with-core-data-tutorial

- My app showcasing many of the things we talked about: https://github.com/lkawka/swift-cheat-sheet/tree/master/PermanentStorage

- Apple guidelines on core data: https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreData/index.html

THANKS FOR LISTENING

THAT'S IT